《汇编语言程序设计》教案

《IBM-PC 汇编语言程序设计

（第 2 版）

沈美明、温冬婵 编著

教案编写时间：**2007** 年 **8** 月 **18** 日

前　　言

1. 汇编语言是计算机能提供给用户的最快而又最有效的语言，也是能够利用计算机所有硬件特性并能直接控制硬件的唯一语言。
2. 汇编语言程序设计是高等院校电子计算机硬、软件及应用专业学生必修的核心课程之一。它不仅是计算机原理、操作系统等其它核心课程的必要先修课，而且对于训练学生掌握程序设计技术、熟悉上机操作和程序调试技术都有重要作用。
3. 本教材共有十一章，其内容安排如下：
   (1). 第一、二章为汇编语言所用的基础知识。
   (2). 第三章详细介绍 80x86 系列 CPU 的指令系统和寻址方式。
   (3). 第四章介绍伪操作、汇编语言程序格式及汇编语言的上机过程。
   (4). 第五、六章说明循环、分支、子程序结构和程序设计的基本方法。
   (5). 第七章说明宏汇编、重复汇编及条件汇编的设计方法。
   (6). 第八章叙述输入/输出程序设计方法，重点说明中断原理、中断过程及中断程序设计方式。
   (7). 第九章说明 BIOS 和 DOS 系统功能调用的使用方法。
   (8). 第十～十一章分别说明图形显示、发声及磁盘文件存储的程序设计方法，同时提供各种程序设计方法和程序实例。

附：教学参考书

1. 沈美明、温冬婵编著，IBM–PC 汇编语言程序设计(第 2 版)，清华大学出版社，2001 年(**教材**)
2. 沈美明、温冬婵编著，IBM–PC 汇编语言程序设计，清华大学出版社，1991 年
3. 沈美明、温冬婵编著，IBM–PC 汇编语言程序设计—例题习题集，清华大学出版社，1991 年 6 月
4. 沈美明、温冬婵、张赤红编著，IBM–PC 汇编语言程序设计—实验教程，清华大学出版社，1992 年
5. 周明德，微型计算机 IBM PC/XT(0520 系列)系统原理及应用(修订版)，清华大学出版社，1991
6. 郑学坚、周斌，微型计算机原理及应用(第二版)，清华大学出版社，1995
7. 王士元、吴芝芳，IBM PC/XT[长城 0520] 接口技术及其应用，南开大学出版社，1990
8. 杨素行，微型计算机系统原理及应用，清华大学出版社，1995
9. 戴梅萼、史嘉权，微型计算机技术及应用—从 16 位到 32 位(第二版)，清华大学出版社，1996
10. 张昆藏，IBM PC/XT 微型计算机接口技术，清华大学出版社，1991
11. 孟绍光，李维星，高档微机组成原理及接口技术(80386/80486/Pentium)，学苑出版社，1993
12. 吴秀清，周荷琴，微型计算机原理与接口技术，中国科学技术大学出版社

目　录

第 1 章 基础知识

【教学目的】

本章内容是本课程的基础，通过本章学习，使学生明确汇编语言程序设计的学科性质、基本内容和学习意义，掌握数制的转换、数据的编码，了解本门课程的教学要求和学习方法。

【重点难点】

二进制数及其与其它数制的转换、补码及其运算。

【课 时 数】

3 学时。

1.1 进位计数制与不同基数的数之间的转换

1.1.1 二进制数

1. 十进制数：(Decimal)(数后面加 D 或省略表示的是十进制数)

(1). 十进制数表示为：$\mathbf{a_n a_{n-1}\cdots a_0.a_{-1}a_{-2}\cdots a_{-m}}$　　其含义如下：

$$N = a_n \bullet 10^n + a_{n-1} \bullet 10^{n-1} + \cdots + a_0 \bullet 10^0 + a_{-1} \bullet 10^{-1} + a_{-2} \bullet 10^{-2} + \cdots + a_{-m} \bullet 10^{-m} = \sum_{i=n}^{-m} a_i \bullet 10^i$$

1). 基数为 10

2). 10 个数码 0 、1、2、3、4、5、6、7、8、9

3). 逢 10 进 1

(2). 权：相应于式中每位数字的 10^k 称为该位数的权。

(3). 数的值：每位数字乘以其权所得到的乘积之和即为该数的值。即如上述多项式展开后所得到的和。

2. r 进制数：

(1). r 进制数表示为：$\mathbf{a_n a_{n-1}\cdots a_0.a_{-1}a_{-2}\cdots a_{-m}}$　　其含义如下：

$$N_r = a_n \bullet r^n + a_{n-1} \bullet r^{n-1} + \cdots + a_0 \bullet r^0 + a_{-1} \bullet r^{-1} + a_{-2} \bullet r^{-2} + \cdots + a_{-m} \bullet r^{-m} = \sum_{i=n}^{-m} a_i \bullet r^i$$

1). 基数为 r

2). r 个数码 0 、1、2、……、 r-1

3). 逢 r 进 1

(2). 权：相应于式中每位数字的 r^k 称为该位数的权。

(3). 数的值：每位数字乘以其权所得到的乘积之和即为该数的值。即如上述多项式展开后所得到的和。

3. 二进制数：(数后面加 B 表示二进制数)

(1). 二进制数表示为：$\mathbf{a_n a_{n-1}\cdots a_0.a_{-1}a_{-2}\cdots a_{-m}}$　　其含义如下：

$$N_2 = a_n \bullet 2^n + a_{n-1} \bullet 2^{n-1} + \cdots + a_0 \bullet 2^0 + a_{-1} \bullet 2^{-1} + a_{-2} \bullet 2^{-2} + \cdots + a_{-m} \bullet 2^{-m} = \sum_{i=n}^{-m} a_i \bullet 2^i$$

1). 基数为 2

2). 2 个数码 0 和 1

3). 逢 2 进 1

(2). 权：相应于式中每位数字的 2^k 称为该位数的权。

(3). 数的值：每位数字乘以其权所得到的乘积之和即为该数的值。即如上述多项式展开后所得到的和。

4. 十六进制数：(Hexadecimal)(数后面加 H 表示十六进制数)

(1). 十六进制数表示为：$\mathbf{a_n a_{n-1}\cdots a_0.a_{-1}a_{-2}\cdots a_{-m}}$　　其含义如下：

$$N_H = a_n \bullet 16^n + a_{n-1} \bullet 16^{n-1} + \cdots + a_0 \bullet 16^0 + a_{-1} \bullet 16^{-1} + a_{-2} \bullet 16^{-2} + \cdots + a_{-m} \bullet 16^{-m} = \sum_{i=n}^{-m} a_i \bullet 16^i$$

1). 基数为 16

2). 16 个数码 0 、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F(或 a ～ f)

3). 逢 16 进 1

(2). 权：相应于式中每位数字的 16^k 称为该位数的权。

(3). 数的值：每位数字乘以其权所得到的乘积之和即为该数的值。即如上述多项式展开后所得到的和。

1.1.2 二进制数和十进制数之间的转换

1. 二进制数转换为十进制数：根据上面的二进制数多项式按权展开求和即为十进制数。

2. 十进制数转换为二进制数

(1). 降幂法：首先写出要转换的十进制数，其次写出所有小于此数的各位二进制权值，然后用要转换的十进制数减去与它最相近的二进制权值，如够减则减去并在相应位记以 1；如不够减则在相应位记以 0 并跳过此位；如此不断反复，直到该数为 0 为止。

(2). 除法

1). 十进制整数转换：采用“除 2 (基)取余”法进行转换，即把十进制整数除以 2，取出余数 1 或 0 作为相应二进制数的最低位，把得到的商再除以 2，再取出余数 1 或 0 作为相应二进制数的次低位。由此类推，继续上述过程直至商为 0 止，最后一次的余数为二进制数的最高位，依次所得到的余数序列就是转换成的二进制数。

2). 十进制小数转换：采用“乘 2 (基)取整”法进行转换，即先将十进制小数乘以 2，取其整数 1 或 0 作为相应二进制小数的最高位，然后将乘积的小数部分再乘以 2，并再取其整数作为次高位。依次重复上述过程，直到小数部分为 0 或达到要求的精度为止。

1.1.3 十六进制数及其与二进制、十进制数之间的转换

1. 十六进制数的表示：(参见 1.1.1 节的第 4 步)以下是几个概念：

(1). 位(bit)。bit(Binary digit 的缩写)是量度信息的最小单位，1 比特为二进制的一位包含的信息量。

(2). 字节(Byte)。作为一个单位来处理的一串二进制数位，通常由 8 位二进制数位组成一个字节。一个字节可以代表一个数字、一个字母或一个特殊符号。也是计算机存储容量的单位。

(3). 字(Word)。在计算机中，一般称两个字节为一个字。

(4). 字长(Word Length)。计算机的每个字所包含的位数称为字长。字长是计算机的一项重要指标。一般都选为字节的整数倍。

2. 十六进制数与二进制数之间的转换

(1). 二进制数转换为十六进制数：从二进制数的小数点位置开始，整数部分向左，小数部分向右，每四位二进制数字为一组用一位十六进制数字表示，不足四位的用 0 补足，就得到一个相应的十六进制数。

(2). 十六进制数转换为二进制数：每一位十六进制数用四位二进制数表示，就形成相应的二进制数了。

3. 十六进制数与十进制数之间的转换

(1). 十六进制数转换为十进制数：根据上面的十六进制数多项式按权展开求和即为十进制数。

(2). 十进制数转换为十六进制数：可类似于十进制数转换为二进制数的降幂法和除法来转换为十六进制数。也可先将十进制数转换为二进制数，再将二进制数转换为十六进制数。

1). 降幂法：首先写出要转换的十进制数，其次写出所有小于此数的各位十六进制权值，然后用要转换的十进制数除以与它最相近的十六进制权值，并在相应位记以十六进制的商；再将余数除以下一位权值，并记下商和余数；如此不断反复，直到该数为 0 为止。

2). 除法：同前，整数部分采用除基(16)取余法，小数部分采用乘基(16)取整法直到所需的精度为止来进行转换。

1.2 二进制数和十六进制数的运算

1.2.1 二进制数的运算

1. 加法规则：0+0=0，0+1=1，1+0=1，1+1=0 (进位 1)

2. 乘法规则：0×0=0，0×1=0，1×0=0，1×1=1

1.2.2 十六进制数的运算：按照逢 16 进 1 的原则进行计算。

1.3 计算机中数和字符的表示

1.3.1 数的补码表示

1. 机器数：在机器中以数值化表示的连同其符号在内的一个数称为机器数。一般用最高有效位来表示符号，正数用 0 表示，负数用 1 表示。

2. 补码表示法

$$[X]_{补}=\begin{cases}|X| & ；\ X\geq 0\\ 2^n-|X| & ；\ X<0\end{cases}$$

3. 补码求法

(1). 正数的补码就是其本身；

(2). 负数的补码 = 其正数的补码按位求反+1。

4. 符号扩展(如将 8 位有符号数扩展成为 16 位有符号数)

(1). 正数的符号扩展是在前面补 0；

(2). 负数的符号扩展是在前面补 1。

5. n 位补码所表示数的范围：$-2^{n-1}\leqslant N\leqslant 2^{n-1}-1$

6. 双字长数或双精度数：在机器里，为了扩大表数范围，可以用两个机器字(高位字和低位字)来表示一个机器数，这种数称为双字长数或双精度数。在 80386 及其后则有 4 字(64 位)。

1.3.2 补码的加法和减法

1. 求补运算：对一个二进制数按位求反后在末尾加 1 的运算称为求补运算。

$$[X]_{补}\xrightarrow{求补}[-X]_{补}\xrightarrow{求补}[X]_{补}$$

2. 补码的加法运算规则

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

3. 补码的减法运算规则

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

1.3.3 无符号整数：把最高有效位也作为数值处理的数称为无符号整数。

1.3.4 字符表示法(ASCII 码)：80x86 机采用目前最常用的美国信息交换标准代码 ASCII 码表示。

1.3.5 BCD 码：4 位二进制数编码的十进制数称为 BCD 码(又叫 8421 码)。BCD 码是无符号整数

1. 压缩 BCD 码：用 4 位二进制数表示一个十进制数位。

2. 非压缩 BCD 码：用 8 位二进制数表示一个十进制数位。

1.4 几种基本的逻辑运算

1.4.1 “与”运算(AND)

$$Y=A\cdot B=A\ \wedge\ B=A\quad AND\quad B$$

1.4.2 “或”运算(OR)

$$Y=A+B=A\ \vee\ B=A\quad OR\quad B$$

1.4.3 “非”运算(NOT)

$$Y=\overline{A}$$

1.4.4 “异或”运算(XOR)

$$Y=A\oplus B=A\forall B=A\quad XOR\quad B$$

所有的逻辑运算都是按位操作的。

第 2 章 80x86 计算机组织

【教学目的】

本章内容也是本课程的基础，通过本章学习，使学生明确汇编语言程序设计所要掌握的存储器的组织方式、CPU 寄存器的结构（编程结构）。

【重点难点】

CPU 寄存器的结构（编程结构）、存储器的组织方式。

【课 时 数】

5 学时。

2.1 80x86 微处理器

1. 80x86 是美国 Intel 公司生产的微处理器系列。

2. 计算机主要由运算器、控制器、存储器和输入输出设备构成。

3. 中央处理器(central processing unit, CPU)：把运算器和控制器集成在一个芯片上，构成中央处理机。80x86 就是这样一组微处理器系列。

4. 很多计算机厂商把微处理器芯片作为中央处理机，再配上存储器、输入输出设备和系统软件等构成微计算机系统。80x86 微处理器系列的主要技术数据名称：

(1). 集成度：晶体管数是指芯片中所包含的晶体管数目，它说明器件的集成度；

(2). 主频是指芯片所用的主时钟频率，它直接影响计算机的运行速度。

(3). 数据总线负责计算机中数据在各组成部分之间的传送。

1). 数据总线宽度是指在芯片内部数据传送的宽度。

2). 外部数据总线宽度则是指芯片内和芯片外交换数据的宽度。

(4). 地址总线宽度是指专用于传送地址的总线宽度，根据这一数值(如为 n)可确定处理机可以访问的存储器的最大范围(寻址空间=2^n)。

1). 字节(byte)：在计算机里，8 个二进制位组成一个字节，一般存储器以字节为存储信息的基本单位，用符号 B 来表示。在存储器里一般用 KB、MB、GB 为存储容量的单位。

2). 1KB=1024B=2^{10}B，1MB=1024KB=2^{20}B，1GB=1024MB=2^{30}B

5. 在计算机里，采用层次结构的存储器组织是解决存储器容量、速度、价格三者矛盾的最有效方法。

(1). 中间层次是主存储器，又称为内存。

(2). 比中间层次速度更高、但容量较小的一层称为高速缓冲存储器(cache)。

(3). 比中间层次速度慢、但容量很大的一层称为外存储器。如磁带、磁盘、光盘等。

6. 提高计算机的工作速度可以说是微处理器芯片发展的核心问题。从 80486 起，把协处理器集成到芯片中的目的也是为了提高浮点处理速度。字长的增加有利于提高计算机解题的精度。

7. 从 80286 开始，在机器的工作方式上，除 8086 提供的实模式外，还增加了保护模式的工作方式。在 80386 中还增加了一种虚拟 8086 的工作模式。

2.2 基于微处理器的计算机系统构成

2.2.1 硬件：是指能看得见、摸得着的物理部件。

1. 组成计算机的三要素：CPU、存储器(memory)和输入/输出(I/O)子系统。用系统总线连接。

(1). 存储器(内存)：计算机的记忆部件。

(2). 中央处理器 CPU：包括运算器和控制器两部分。

1). 运算器执行所有的算术和逻辑运算指令；

2). 控制器则负责全机的控制工作。

(3). I/O 子系统：包括 I/O 设备及大容量存储器两类外部设备。

1). I/O 设备是指负责与计算机的外部世界通信用的输入、输出设备；

2). 大容量存储器是指可存储大量信息的外部存储器。

2. 系统总线：把 CPU、存储器和 I/O 设备连接起来，用来传送各部件之间的信息。系统总线的动作由总线控制逻辑负责指挥。

(1). 数据总线：传送信息；

(2). 地址总线：指出信息的来源和目的地；
(3). 控制总线：规定总线的动作。

2.2.2 软件：为运行、管理和维护计算机而编制的各种程序的总和。分为系统软件和用户软件两大类。

1. 系统软件：由计算机生产厂家提供给用户的一组程序。其核心是操作系统。它包括：
(1). 操作系统的主要部分是常驻监督程序(monitor)。
(2). I/O 驱动程序：对 I/O 设备进行控制和管理。
(3). 文件管理程序：处理存储在外存储器中的大量信息。
(4). 文本编辑程序：建立、输入或修改文本，并存入内存储器或大容量存储器中(如 EDIT 等)。
(5). 翻译程序(translator)
1). 汇编程序：把由用户编制的汇编语言源程序翻译成机器语言目标程序。
a. 汇编语言：一种与机器语言几乎一 一对应的符号语言，但在书写时使用由字符串组成的助记符。
b. 机器语言：由二进制代码组成的语言。
c. 指令：计算机能识别并能直接加以执行的二进制的语句。
2). 编译程序：把高级语言源程序翻译成机器语言程序的系统程序。
3). 解释程序：对高级语言一边解释一边执行的翻译程序。
(6). 连接程序(linker)：把要执行的程序与库文件或其他已经翻译的子程序连接在一起，形成机器能执行的程序。
(7). 装入程序(loader)：把程序从外存储器装入内存储器，以便机器运行。
(8). 调试程序(debug)：系统提供给用户能监督和控制用户程序的一种工具。
(9). 系统程序库和用户程序库：各种标准程序、子程序及一些文件的集合。
2. 用户软件：用户自行编制的各种应用程序。

2.3 中央处理机

2.3.1 中央处理机 CPU 的组成

1. CPU 的任务是执行存放在存储器里的指令序列。
2. CPU 芯片中除高速缓冲存储器之外，主要由以下三部分组成：
(1). 算术逻辑部件(arithmetic logic unit，ALU)：用来进行算术和逻辑运算。
(2). 控制逻辑：负责对全机的控制工作。
(3). 工作寄存器：用来存放计算过程中所需要的或所得到的各种信息。

2.3.2 80x86 寄存器组

寄存器可以分为程序可见的寄存器和程序不可见的寄存器两大类。

① 程序可见的寄存器是指在汇编语言程序设计中用到的寄存器，分为通用寄存器、专用寄存器和段寄存器 3 类。

② 程序不可见的寄存器是指一般应用程序设计中不用而由系统所用的寄存器。

1. 通用寄存器
(1). 数据寄存器：AX、BX、CX、DX，四个 16 位通用寄存器，用来暂时存放计算过程中所用到的操作数、结果和其他信息。既可以以字形式(如 AX)也可以以字节形式(如 AH、AL)访问。
1). AX(accumulator)：累加器，算术运算的主要寄存器。所有的 I/O 指令都使用这一寄存器与外部设备传送信息。
2). BX(base)：通用寄存器，在计算存储器地址时常用作基址寄存器。
3). CX(count)：通用寄存器，在循环和串操作指令中用作隐含的计数器。
4). DX(data)：通用寄存器，在作双字长运算时把 DX 和 AX 合在一起存放一个双字长数，DX 用来存放高位字。对某些 I/O 操作，DX 用于对 I/O 端口的寄存器间接寻址。
(2). 指针及变址寄存器：SP、BP、SI、DI，四个 16 位寄存器。
1). SP：堆栈指针寄存器。
2). BP：基址指针寄存器。
3). SI：源变址寄存器。
4). DI：目的变址寄存器。
(3). 对于 80386 及其后继机型则是 32 位的通用寄存器，包括 EAX、EBX、ECX、EDX、ESP、EBP、

EDI 和 ESI。这些寄存器都可以存放数据，也可以当 32 位的地址寄存器使用。

2. 专用寄存器：IP、SP、FLAGS，3 个 16 位寄存器。

(1). IP：指令指针寄存器。存放代码段中的偏移地址。80386 及其后继机型则是 EIP。

(2). SP：堆栈指针寄存器，指示栈顶的偏移地址。80386 及其后继机型则是 ESP。

(3). FLAGS：标志寄存器，又称为程序状态字寄存器(program status word, PSW)。由条件码标志(flag)、控制标志和系统标志构成。80386 及其后继机型则是 EFLAGS。8086/8088 的 FLAGS 如下所示：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

1). 条件码：用来记录程序中运行结果的状态信息，它们是根据有关指令的运行结果由 CPU 自动设置的。由于这些状态信息往往作为后续条件转移指令的转移控制条件，所以称为条件码。

a. OF：溢出标志(overflow flag)。结果溢出 OF=1，否则 OF=0。

b. SF：符号标志(sign flag)。结果为负 SF=1，否则 SF=0。

c. ZF：零标志(zero flag)。结果为 0 时 ZF=1，否则 ZF=0。

d. CF：进位标志(carry flag)。最高有效位有进位或借位 CF=1，否则 CF=0。

e. AF：辅助进位标志或半进位标志(auxiliary carry flag)。半个字节产生的进位或借位时 AF=1，否则 AF=0。

f. PF：奇偶标志(parity flag)。结果低 8 位中所含的 1 的个数为偶数时 PF=1，否则 PF=0。

2). 控制标志位：为方向标志(direction flag, DF)，在串处理指令中控制处理信息的方向用。

a. DF=1：变址寄存器 SI 和 DI 减量，使串处理从高地址向低地址方向处理。

b. DF=0：变址寄存器 SI 和 DI 增量，使串处理从低地址向高地址方向处理。

3). 系统标志位：可以用于 I/O、可屏蔽中断、程序调试、任务切换和系统工作方式等的控制。

a. TF：陷阱标志(trap flag, TF)，又称单步标志。用于调试时的单步方式操作。TF=1 时每条指令执行完后产生陷阱(中断)，TF=0 时不产生陷阱。

b. IF：中断标志(interrupt flag, IF)。IF=1 时允许可屏蔽中断请求，IF=0 时禁止可屏蔽中断。

c. I/O 特权级(I/O privilege level, IOPL)，在保护模式下，用于控制对 I/O 地址空间的访问。

(4). 标志位的测试：用调试程序 DEBUG 测试。测试含义见书 24 页表 2.2 所示。

3. 段寄存器：是一种专用寄存器，它们专用于存储器寻址，用来直接或间接地存放段地址。在 80286 以前的处理器中只有 CS、DS、SS、ES 四个 16 位寄存器。从 80386 起增加了 FS 和 GS 两个附加的数据段寄存器。

2.4 存储器

2.4.1 存储单元的地址和内容：

1. 字节：每 8 位二进制数组成一个字节(Byte)。在存储器里是以字节为单位存储信息。

2. 字：16 位二进制数(2 个字节)组成一个字。分成低位字节和高位字节存储。

3. 存储单元的地址：为了正确的存放或取得信息，每一个字节单元给以一个唯一的存储器地址，称为物理地址。以字节为单位给存储单元按二进制从 0 开始顺序进行的编号。它是无符号的二进制整数，书写格式为十六进制数。

(1). 选址能力：若干位(如 x 位)二进制数(相当于若干根地址线)所能选择的最大存储单元的地址数(2^x)。

(2). 1K=1024=2^{10}。所以 16 位二进制数的选址能力为 2^{16}=64KB。用 0～FFFFH 表示。IBM PC 机(8086/8088 CPU)的选址能力为 2^{20}=1MB。

4. 存储单元的内容：该存储单元存放的信息。存储器的内容取之不尽。

(1). 字节的存储：直接存入某存储单元。

(2). 字的存储：一个字存入存储器要占用两个单元。存放时，低位字节存入低地址单元，高位字节存入高位地址单元。字单元的地址用它的低地址表示，应尽量为偶数地址。

(3). 如果用 X 表示某存储单元的地址，则 X 单元的内容可以表示为(X)；假如 X 单元中存放着 Y，而 Y 又是一个地址，则可用(Y)=((X))来表示 Y 单元的内容。

(4). 存储器的内容取之不尽。

2.4.2 实模式存储器寻址

80x86 中除 8086/8088 只能在实模式下工作外，其他微处理器均可在实模式或保护模式下工作。实模式就是为 8086/8088 而设计的工作方式，实模式下允许的最大寻址空间为 1MB。它要解决在 16 位字长的机器里怎么提供 20 位地址的问题，而解决的办法是采用存储器地址分段的方法。

1. 存储器地址的分段

(1). 段：程序员在编制程序时要把存储器划分成段，在每个段内地址空间是线性增长的。段的最大长度为 64KB，以便能用 16 位二进制数表示段内地址。

1). 小段：从 0 地址开始，每 16 字节为一小段。小段的首地址用 16 进制数表示时其最低位为 0。

2). 段的起始地址：段不能起始于任意地址，而必须从任一小段(paragraph)的首地址开始。

(2). 物理地址、段地址、偏移地址

1). 物理地址(PA)：在 1MB 的存储器里，每一个存储单元都有一个唯一的 20 位地址，称为该存储单元的物理地址(20 位)。20 位物理地址由 16 位段地址和 16 位偏移地址组成。

2). 段地址：段起始地址的高 16 位值(16 位)，低 4 位固定为 0 被省去。

3). 偏移地址(偏移量 EA)：指在段内相对于段起始地址的偏移值(16 位)。

4). 物理地址 = 段地址×10H + 偏移地址。用**段地址：偏移地址**表示。

2. 段寄存器：在 8086~80286 中，有 4 个专门存放段地址的寄存器。

(1). 代码段 CS：存放当前正在运行的程序。

(2). 数据段 DS：存放当前运行程序所用的数据。

(3). 堆栈段 SS：定义堆栈的所在区域。

(4). 附加段 ES：附加的数据段。在 80386 及其后又增加了 FS 和 GS 两个附加数据段寄存器。

除非专门指定，一般情况下，各段在存储器中的分配是由操作系统负责的。而且允许段重叠。如果段超过 64KB 则将其作为 2 个以上的段，并动态修改段地址。

2.4.3 保护模式存储器寻址

从 80286 起，就引出了保护模式的存储器寻址，其直接原因：首先是实模式的寻址空间为 1MB，这不能满足 80286 的 16MB 和 80386 等的 4GB 或更多的地址空间寻址。其次是为了使微机系统能支持多任务处理。在系统支持多任务功能的同时，系统也支持了虚拟存储器特性。

1. 逻辑地址

(1). 在实模式下逻辑地址即为**段地址：偏移地址**。物理地址 = 段地址×10H + 偏移地址

(2). 在保护模式下逻辑地址则由**选择器和偏移地址**两部分组成。选择器存放在段寄存器中，但它不能直接表示段基地址，而由操作系统通过一定的方法取得段基地址，再和偏移地址相加，从而求得所选存储单元的物理地址。

2. 描述符

(1). 描述符用来描述段的大小、段在存储器中的位置及其控制和状态信息，它由基地址、界限、访问权和附加字段四部分组成。描述符的内容是由系统设置的，而不是由用户建立的。

1). 基地址(base)部分用来指定段的起始地址；

2). 界限(limit)部分存放着该段的段长度；

3). 访问权(access rights)部分用来说明该段在系统中的功能，并给出该段的一些控制信息；

4). 附加字段部分在 386 及其后继机型中存在，它用来表示该段的一些属性。

(2). 系统按选择器的内容，根据指定的途径可以找到所选段对应的描述符，从而可以根据其给出的基地址和界限值，确定所要找的存储单元所在的段，再加上逻辑地址中指定的偏移地址，就可以找到相应的存储单元。

2.5 外部设备

1. 端口寄存器：外部设备与主机(CPU 和存储器)的通信是通过接口进行的。每个接口包括一组寄存器称为端口寄存器。

(1). 数据寄存器：存放要在外设和主机之间传送的数据。

(2). 状态寄存器：保存外设或接口的状态信息。

(3). 命令寄存器：CPU 发给外设或接口的控制命令通过它传送。

2. 端口地址(端口号)：主机给外设中的每个寄存器定义一个二进制数的编号即为端口地址。其地址空间独立于内存地址之外可达 64KB。其范围为 0000～FFFFH。

3. 主机与外设交换信息：通过输入/输出指令完成。

4. BIOS、DOS 功能调用：操作系统提供的中断服务子程序。

(1). BIOS 存放在机器的 ROM 中，比 DOS 的层次还要低。

(2). DOS 功能调用是 DOS 的一个组成部分，在开机时由操作系统从系统磁盘装入存储器。

第 3 章 80x86 的指令系统和寻址方式

【教学目的】

本章内容是本课程的重点，通过本章学习，使学生掌握 80x86 系列 CPU（特别是 8086CPU）的指令系统和寻址方式。对传送类指令、算术运算类指令、控制转移类指令要非常熟悉。

【重点难点】

传送类指令、算术运算类指令、控制转移类指令、存储器寻址方式、段内直接寻址的地址位移量。

【课 时 数】

12 学时。

1. 指令系统：计算机能够提供给用户的一组指令集即为该计算机的指令系统。

2. 指令的组成：计算机中的指令由操作码字段和操作数字段组成。

(1). 操作码字段：指示计算机所要执行的操作。

(2). 操作数字段：指出在指令执行操作的过程中所需要的操作数。

3. 指令的格式：**操作码 [操作数 [，操作数] [，操作数]]**。8086/8088 CPU 规定操作数不能超过两个。但在 80286 及其后的 CPU 可以使用 3 操作数指令。

4. 源操作数和目的操作数：使用两地址指令的两个操作数分别称为源操作数和目的操作数。

5. 操作数的表示方法使用的是寻址方式。寻址方式是规定寻找操作数的方法。

6. 汇编语言：符号语言。用助记符来表示操作码，用符号或符号地址来表示操作数或操作数地址。它与机器指令一 一对应。

3.1 80x86 的寻址方式

3.1.1 与数据有关的寻址方式

这种寻址方式用来确定操作数地址从而找到操作数。

1. 立即数寻址方式：操作数直接存放在指令中，紧跟在操作码之后，这种操作数称为立即数。它作为指令的一部分存放在代码段里。如：MOV AL, 6；MOV AX, 300AH；MOV EAX, 2030300AH

2. 寄存器寻址方式：操作数在寄存器中，指令指定寄存器号。如：MOV AL, AH

3. 直接寻址方式：操作数的有效地址只包含位移量一种成份，其值就存放在代码段中的指令的操作码之后。位移量的值即操作数的有效地址 EA。如：MOV AX, VALUE； MOV AX, [2000H]

(1). 由此及往下的各种寻址方式的操作数都在除代码段以外的存储区中。用方括号(“[ ]”)括起来的为存储器操作数。寄存器名称外加小括号“( )”表示是该寄存器的内容。

(2). 有效地址 (effective address, EA)：即操作数的偏移地址。自此开始的寻址方式即为求得有效地址(EA)的不同途径。有效地址的计算可以用下式表示：

EA＝基址＋(变址×比例因子)＋位移量

有效地址可以由以下四种成分组成：

1). 位移量(displacement)是存放在指令中的一个 8 位、16 位或 32 位的数，但它不是立即数，而是一个地址。

2). 基址(base)是存放在基址寄存器中的内容。它是有效地址中的基址部分，通常用来指向数据段中数组或字符串的首地址。

3). 变址(index) 是存放在变址寄存器中的内容。它通常用来访问数组中的某个元素或字符串中的某个字符。

4). 比例因子(scale factor)是 386 及其后继机型新增加的寻址方式中的一个术语，其值可为 1、2、4 或 8。在寻址中可用变址寄存器的内容乘以比例因子来取得变址值。

(3). 段跨越前缀：80x86 允许数据存放在数据段以外的段中，应在指令中用该段寄存器加冒号(“：”)即段跨越前缀来指定该段。只要有 BP 则隐含的段寄存器为 SS。否则隐含的段寄存器为 DS。如：MOV AX, ES: VALUE。但是在以下三种情况下，不允许使用段跨越前缀，它们是：

1). 串处理指令的目的串必须使用 ES 段；

2). PUSH 指令的目的和 POP 指令的源必须用 SS 段；

3). 指令代码必须存放在 CS 段中。

(4). 80x86 CPU 中为了使指令字不要过长，规定双操作数指令的两个操作数中，只能有一个使用存储器寻址方式，这就是一个变量常常先要送到寄存器的原因。

4. 寄存器间接寻址方式：操作数的有效地址只包含基址寄存器或变址寄存器内容一种成份。有效地址就在某个寄存器中，而操作数则在存储器中。可使用段跨越前缀。如：MOV AX, ES: [BX]

5. 寄存器相对寻址方式(或称直接变址寻址方式)：操作数的有效地址为基址寄存器或变址寄存器的内容和指令中指定的位移量之和，有效地址由两部分组成。可使用段跨越前缀(又称为段超越)。如：MOV ES: STRING[SI], DL

$$EA=\begin{Bmatrix}(BX)\\(BP)\\(DI)\\(SI)\end{Bmatrix}+\begin{Bmatrix}8位位移量\\16位位移量\end{Bmatrix}$$

对 80386 及以后机型。寄存器为 EAX、EBX、ECX、EDX、ESI、EDI、ESP、EBP。位移量可增加到 32 位的位移量(无 16 位位移量)。下同。

6. 基址变址寻址方式：操作数的有效地址 EA 是一个基址寄存器和一个变址寄存器的内容之和。可使用段跨越前缀。如：MOV AX, ES: [BX][SI]

$$EA=\begin{Bmatrix}(BX)\\(BP)\end{Bmatrix}+\begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}$$

7. 相对基址变址寻址方式：操作数的有效地址 EA 是一个基址寄存器与一个变址寄存器的内容和指令中指定的位移量之和。可使用段跨越前缀。如：MOV ES: MASK[BX][SI], AX

$$EA=\begin{Bmatrix}(BX)\\(BP)\end{Bmatrix}+\begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}+\begin{Bmatrix}8位位移量\\16位位移量\end{Bmatrix}$$

8. 比例变址寻址方式：操作数的有效地址 EA 是变址寄存器的内容乘以指令中指定的比例因子再加上位移量之和。如：MOV MASK[ESI*4], EAX

9. 基址比例变址寻址方式：操作数的有效地址 EA 是变址寄存器的内容乘以指令中指定的比例因子再加上基址寄存器的内容之和。如：MOV [EAX][ESI*4], EAX

10. 相对基址比例变址寻址方式：操作数的有效地址 EA 是变址寄存器的内容乘以指令中指定的比例因子，加上基址寄存器的内容，再加上位移量之和。如：MOV TABLE[EAX][ESI*4], EAX

11. 端口(输入/输出)寻址方式：一个操作数必须为 AX 或 AL 或 EAX，另一个用端口直接寻址方式(端口号＜256)或端口间接寻址方式(DX 的内容为端口号)。如：IN AL, 0ADH ； OUT DX, AX

3.1.2 与转移地址有关的寻址方式

这种寻址方式用来确定转移指令及 CALL 指令的转向地址。

1. 段内直接寻址：转向的有效地址是当前 IP 内容和指令中指定的 8 位或 16 位位移量之和。(操作数 OPR 采用相对寻址方式。) 当它用于条件转移指令时，位移量只能是 8 位(386 及其后继机型条件转移指令的位移量可为 8 位或 32 位)。

$$(IP)\leftarrow EA=(IP)+\begin{Bmatrix}8位位移量，如：JMP\ SHORT\ \ OPR & ；段内直接短程转移\\16位位移量，如：JMP\ NEAR\ PTR\ OPR & ；段内直接近程转移\end{Bmatrix}$$

2. 段内间接寻址：转向的有效地址是一个寄存器或是一个存储单元的内容。(操作数 OPR 采用除立即数以外的任一数据寻址方式。)

$$(IP)\leftarrow EA=\begin{Bmatrix}寄存器内容，如：JMP\ BX\ \ 或\ \ JMP\ ECX & ；OPR为BX或ECX\\存储器内容，如：JMP\ WORD\ PTR\ [存储器寻址方式] & ；OPR为存储器\end{Bmatrix}$$

3. 段间直接寻址：指令中直接提供了转向的段地址和偏移地址。(操作数 OPR 采用立即数寻址方式。)

(IP)←EA=OFFSET OPR ；OPR 的偏移地址→(IP)

(CS)←SEG OPR ；OPR 的段地址→(CS)

4. 段间间接寻址：用存储器中两个相连字来取代 IP 和 CS 的内容。(操作数 OPR 采用存储器寻址方式。)

(IP)←EA=存储器中双字单元的低字内容

(CS)←EA+2=存储器中双字单元的高位字内容

附：书上 3.1.2 所用到的操作符意义：

1. SHORT：属性操作符，表示段内短程转移。

2. PTR：属性操作符，建立一个符号地址(取后面内容的地址)。

3. NEAR：类型操作符，距离类型，段内近程。

4. FAR：类型操作符，距离类型，段间远程。
5. WORD：类型操作符，数据类型，字。
6. DWORD：类型操作符，数据类型，双字。
7. NEAR PTR：取段内近程地址值操作符。
8. FAR PTR：取段间远程地址值操作符。
9. WORD PTR：取字长地址值操作符。
10. DWORD PTR：取双字长地址值操作符。

3.2 程序占有的空间和执行时间

1. 80x86 的机器指令是可变字节指令，即不同指令或不同寻址方式的机器指令长度不同。程序量越大，占有的存储空间也越大。
2. 当程序在计算机上运行时，访问存储器取得操作数或者存放结果需要时间，运算器执行指令也需要时间。
3. 完成同样功能的不同程序，可能在占有存储空间和执行时间上有很大差别。程序员在编制程序时，应尽量考虑节省程序所占用的空间和所使用的时间。

3.3 80x86 的指令系统

3.3.1 数据传送指令：负责把数据、地址或立即数传送到寄存器或存储单元中。

1. 通用数据传送指令

(1). MOV——传送指令

指令格式：MOV DST，SRC ；(DST)←(SRC)。DST 表示目的操作数, SRC 表示源操作数

说 明：①.DST 为除 CS 外的各寄存器寻址方式或任意存储器寻址方式。SRC 为任意数据寻址方式。

②.DST、SRC 不能同时为存储器寻址方式，也不能同时为段寄存器寻址方式，而且在 DST 为段寄存器时，SRC 不能为立即数。

③.MOV 指令不影响标志位。

(2). MOVSX——带符号扩展传送指令(386 及其后继机型可用)

指令格式：MOVSX DST，SRC ；(DST)←符号扩展(SRC)

说 明：①.DST 必须为 16 位或 32 位寄存器。SRC 为 8 位或 16 位的寄存器或存储单元的内容。传送时把源操作数符号扩展送入目的寄存器。

②.MOVSX 指令不影响标志位。

(3). MOVZX——带零扩展传送指令(386 及其后继机型可用)

指令格式：MOVZX DST，SRC ；(DST)←零扩展(SRC)

说 明：①.DST 必须为 16 位或 32 位寄存器。SRC 为 8 位或 16 位的寄存器或存储单元的内容。传送时把源操作数零扩展送入目的寄存器。

②.MOVZX 指令不影响标志位。

(4). PUSH——进栈指令

指令格式：PUSH SRC ；16 位指令：(SP)←(SP) –2 ((SP)+1,(SP))←(SRC)

32 位指令：(ESP)←(ESP) –4 ((ESP)+3, (ESP)+2, (ESP)+1, (ESP))←(SRC)

说 明：①.堆栈：计算机开辟的以“后进先出”方式工作的存储区。它必须存在于堆栈段中，只有一个出入口，所以只有一个堆栈指针 SP 或 ESP。SP 或 ESP 的内容在任何时候都指向当前的栈顶。

②.8086 中的 SRC 不能为立即数寻址方式。286 及其后继机型可用立即数寻址方式。

③.PUSH 指令不影响标志位。

(5). POP——出栈指令

指令格式：POP DST ；16 位指令：(DST)←((SP)+1,(SP)) (SP)←(SP)+2

32 位指令：(DST)←((ESP)+3, (ESP)+2, (ESP)+1, (ESP)) (ESP)←(ESP)+4

说 明：①.DST 为除立即数及 CS 寄存器以外的任意数据寻址方式。

②.POP 指令不影响标志位。

(6). PUSHA/ PUSHAD——所有寄存器进栈指令

指令格式：PUSHA ；16 位通用寄存器依次进栈，进栈次序为：AX、CX、DX、BX、指令执行前的 SP、BP、SI、DI。指令执行后(SP)←(SP) –16 仍指向栈顶。

指令格式：PUSHAD ；32 位通用寄存器依次进栈，进栈次序为：EAX、ECX、EDX、EBX、指令执行前的 ESP、EBP、ESI、EDI。指令执行后(SP)←(SP) –32 仍指向栈顶。32 位地址时用 ESP。

(7). POPA/ POPAD——所有寄存器出栈指令

指令格式：POPA ；16 位通用寄存器依次出栈，出栈次序为：DI、SI、BP、SP、BX、DX、CX、AX。指令执行后(SP)←(SP) +16 仍指向栈顶。注意 SP 内容并未恢复。

指令格式：POPAD ；32 位通用寄存器依次出栈，出栈次序为：EDI、ESI、EBP、ESP、EBX、EDX、ECX、EAX。指令执行后(SP)←(SP) +32 仍指向栈顶。注意 ESP 内容并未恢复。32 位地址时用 ESP。

说　　明：①.PUSHA 和 POPA 可用于 286 及其后继机型。PUSHAD 和 POPAD 可用于 386 及其后继机型。

②.PUSHA、POPA、PUSHAD、POPAD 指令均不影响标志位。

(8). XCHG——交换指令

指令格式：XCHG　OPR1，OPR2　；(OPR1)←→(OPR2)。其中 OPR 表示操作数

说　　明：①.OPR1、OPR2 为除段寄存器以外的各寄存器寻址方式或任意存储器寻址方式。

②.OPR1、OPR2 不能同时为存储器寻址方式。

③.XCHG 指令不影响标志位。

2. 累加器专用传送指令

(1). IN——输入指令

长格式为：IN　AL，PORT(字节)　；(AL)←(PORT) (字节)

IN　AX，PORT(字)　；(AX)←(PORT+1,PORT) (字)

IN　EAX，PORT(双字)；(EAX)←(PORT+3, PORT+2, PORT+1,PORT) (双字)

短格式为：IN　AL，DX(字节)　；(AL)←((DX)) (字节)

IN　AX，DX(字)　；(AX)←((DX)+1,(DX)) (字)

IN　EAX，DX(双字)　；(EAX)←((DX)+3, (DX)+2, (DX)+1, (DX)) (双字)

说　　明：①.80x86 CPU 规定只能用低 16 位地址总线(A_{15}～A_0)来寻址外部设备，因此外部设备最多可有 65536 个 I/O 端口，端口地址为(0～FFFFH)。

②.长格式只适用于端口(PORT)号≤255 (FFH)。

③.短格式适用于任意端口号(0～FFFFH)。但只能用 DX 寄存器对端口地址进行间接寻址。

④.IN 指令不影响标志位。

(2). OUT——输出指令

长格式为：OUT　PORT, AL(字节)　；(PORT)←(AL) (字节)

OUT　PORT, AX(字)　；(PORT+1,PORT)←(AX) (字)

OUT　PORT, EAX(双字)；(PORT+3, PORT+2, PORT+1,PORT)←(EAX) (双字)

短格式为：OUT　DX，AL(字节)　；((DX))←(AL) (字节)

OUT　DX，AX(字)　；((DX)+1,(DX)←(AX) (字)

OUT　DX，EAX(双字)　；((DX)+3, (DX)+2, (DX)+1,(DX)←(EAX) (双字)

说　　明：①.长格式和短格式的规定与 IN 指令相同。

②.OUT 指令不影响标志位。

(3). XLAT——换码指令

指令格式：XLAT　OPR　；16 位指令：(AL)←((BX)+(AL))

32 位指令：(AL)←((EBX)+(AL))

XLAT　；上式的简写，OPR 为阅读程序用的表格首地址。

说　　明：①.在使用这条指令前，应先建立一个字节表格，表格的首地址应提前存入 BX 寄存器，需要转换的代码应该是相对于表格首地址的位移量也应提前存入 AL 寄存器中。表格的内容则是所要换取的代码，该指令执行后就可在 AL 中得到转换后的代码。

②.XLAT 指令不影响标志位。

3. 地址传送指令

(1). LEA——有效地址(EA)送寄存器指令

指令格式：LEA REG，SRC ；(REG)←SRC

说 明：①.指令把源操作数(只能是存储器寻址方式)指定的有效地址送到指令指定的 16 位或 32 位寄存器(REG)中(但不能是段寄存器)。

②.LEA 指令不影响标志位。

(2). LDS、LES、LFS、LGS、LSS——地址指针送寄存器和相应段寄存器指令，以 LDS 为例

指令格式：LDS REG，SRC ；(REG)←(SRC)，(DS)←(SRC+2)或(DS)←(SRC+4)

说 明：①.该组指令的源操作数只能用存储器寻址方式，根据任一种存储器寻址方式找到一个存储单元。

②.该组指令不影响标志位。

4. 标志寄存器传送指令

(1). LAHF——标志送 AH 指令

指令格式：LAHF ；(AH)←(FLAGS 的低位字节)

(2). SAHF——AH 送标志寄存器指令

指令格式：SAHF ；(FLAGS 的低位字节)←(AH)

(3). PUSHF/PUSHFD——标志进栈指令

指令格式：PUSHF ；(SP)←(SP) −2，((SP)+1,(SP))←(FLAGS)

PUSHFD ；(ESP)←(ESP) −4，((ESP)+3, (ESP)+2, (ESP)+1, (ESP))←(EFLAGS AND 0FCFFFFH)(清除 VM 和 RF 位)

(4). POPF/POPFD——标志出栈指令

指令格式：POPF ；(FLAGS)←((SP)+1,(SP))，(SP)←(SP)+2

POPFD ；(EFLAGS)←((ESP)+3, (ESP)+2, (ESP)+1, (ESP))，(ESP)←(ESP) −4

说 明：这组指令中 LAHF、PUSHF/PUSHFD 不影响标志位。但 POPFD 指令不影响 VM，RF，IOPL，VIF 和 VIP 的值。

5. 类型转换指令

(1). CBW——字节转换为字指令

指令格式：CBW ；(AH)←AL 内容的符号位，形成 AX 中的字。

(2). CWD/CWDE——字转换为双字指令

指令格式：CWD ；(DX)←AX 内容的符号位，形成 DX：AX 中的双字。.

指令格式：CWDE ；AX 内容的符号扩展到 EAX 的高位，形成 EAX 中的双字。

(3). CDQ——双字转换为 4 字指令

指令格式：CDQ ；(EDX)←EAX 内容的符号位，形成 EDX：EAX 中的 4 字。.

(4). BSWAP——字节交换指令

指令格式：BSWAP reg32 ；使指令指定的 32 位寄存器的字节次序变反。具体操作为：1、4 字节互换，2、3 字节互换。

说 明：该指令只能用于 486 及其后继机型。reg32 指 32 位寄存器。

3.3.2 算术指令

80x86 的算术运算指令包括二进制运算及十进制运算指令。算术指令用来执行算术运算，它们中有双操作数指令，也有单操作数指令。双操作数指令的两个操作数中除源操作数为立即数的情况外，必须有一个操作数在寄存器中。单操作数指令不允许使用立即数寻址方式。

1. 加法指令

(1). ADD——加法指令

指令格式：ADD DST，SRC ；(DST)←(DST)+(SRC)

(2). ADC——带进位加法指令

指令格式：ADC DST，SRC ；(DST)←(DST)+(SRC)+CF

(3). INC——加 1 指令

指令格式：INC OPR ；(OPR)←(OPR)+1

说 明：①.以上指令除 INC 不影响 CF 标志外，它们都影响条件标志位。

②.OF 是有符号数的溢出，CF 是无符号数的溢出。但 CF 可作为多位运算的进位标志。

(4). XADD——交换并相加指令

指令格式：XADD DST，SRC ；TEMP←(DST)+(SRC)，(SRC)←(DST)，(DST)←TEMP

说 明：①.该指令只能用于 486 及其后继机型。

②.源操作数只能用寄存器寻址方式，目的操作数则可用寄存器或任一种存储器寻址方式。指令可作双字、字或字节运算。

2. 减法指令

(1). SUB——减法指令

指令格式：SUB DST，SRC ；(DST)←(DST)－(SRC)

(2). SBB——带借位减法指令

指令格式：SBB DST，SRC ；(DST)←(DST)－(SRC)－CF

(3). DEC——减 1 指令

指令格式：DEC OPR ；(OPR)←(OPR)－1

(4). NEG——求补指令

指令格式：NEG OPR ；(OPR)← －(OPR) 即(OPR)←0－(OPR)

(5). CMP——比较指令

指令格式：CMP OPR1，OPR2 ；(OPR1) - (OPR2)，运算后根据结果影响标志

说 明：①.以上指令除 DEC 不影响 CF 标志外，它们都影响条件标志位。

②.OF 是有符号数的溢出，CF 是无符号数的溢出。但 CF 可作为多位运算的借位标志。

(6). CMPXCHG——比较并交换指令

指令格式：CMPXCHG DST, SRC ；累加器 AC 与 DST 相比较，若(AC)=(DST)，则 ZF←1，(DST)←(SRC)；否则 ZF←0，(AC)←(DST)

说 明：①.该指令只能用于 486 及其后继机型。

②.源操作数只能用 8 位、16 位或 32 位寄存器寻址，目的操作数则可用寄存器或任一种存储器寻址方式。该指令对其他标志的影响与 CMP 相同。

(7). CMPXCHG8B——比较并交换 8 字节指令

指令格式：CMPXCHG8B DST ；EDX, EAX 与 DST 比较，若(EDX, EAX)=(DST)，则 ZF←1，(DST)←(ECX, EBX)；否则 ZF←0，(EDX, EAX)←(DST)

说 明：①.该指令只能用于 Pentium 及其后继机型。

②.源操作数为存放于 EDX，EAX 中的 64 位字，目的操作数可用存储器寻址方式确定一个 64 位字。该指令只影响 ZF 标志。

3. 乘法指令

(1). MUL——无符号数乘法指令

指令格式：MUL SRC ；字节操作：(AX)←(AL)×(SRC)

字 操 作：(DX, AX)←(AX)×(SRC)

双字操作：(EDX, EAX)←(EAX)×(SRC)

(2). IMUL——带符号数乘法指令

指令格式：IMUL SRC ；与 MUL 相同，但必须是带符号数，而 MUL 是无符号数

说 明：①.在乘法指令中目的操作数必须是累加器，字运算为 AX，字节运算为 AL，双字运算为 EAX，指令中不写出。SRC 不能用立即数。

②.乘法指令对除 CF 和 OF 以外的条件码无定义。(无定义是指该标志位不确定。)

1). IMUL——在 80286 及其后继机型中的双操作数的带符号数乘法指令

指令格式：IMUL REG，SRC ；字 操 作：(REG16)←(REG16)×(SRC)

双字操作：(REG32)←(REG32)×(SRC)

说 明：①.目的操作数必须是 16 位或 32 位寄存器，而源操作数则可用任一种寻址方式取得和目的操作数长度相同的数；OF=1 时溢出。

②.如果源操作数为立即数时，除相应地用 16 位或 32 位立即数外，指令中也可指定 8 位立即数，在运算时机器会自动把该数符号扩展成与目的操作数长度相同的数。

2). IMUL——在 80286 及其后继机型中的三操作数的带符号数乘法指令

指令格式：IMUL REG, SRC, IMM ；字 操 作：(REG16)←(SRC)×IMM

双字操作：(REG32)←(SRC)×IMM

说 明：①.目的操作数必须是 16 位或 32 位寄存器，而源操作数则可用除立即数以外的任一种寻址方式取得和目的操作数长度相同的数；OF=1 时溢出。

②.IMM 表示立即数，它可以是 8、16 或 32 位数，但其长度必须与目的操作数一致，如长度为 8 位时，运算时将符号扩展成与目的操作数长度相同的数。

4. 除法指令

(1). DIV——无符号数除法指令

指令格式：DIV SRC ；字节操作：(AL)←(AX)/(SRC)，(AH)←(AX)%(SRC)

字 操 作：(AX)←(DX,AX)/(SRC)，(DX)←(DX,AX)%(SRC)

双字操作：(EAX)←(EDX,EAX)/(SRC)，(EDX)←(EDX,EAX)%(SRC)

(2). IDIV——带符号数除法指令

指令格式：IDIV SRC ；与 DIV 相同，但操作数必须是带符号数，商和余数也都是带符号数，且余数的符号与被除数的符号相同

说 明：①.在除法指令中目的操作数必须是 AX 或 DX，AX，指令中不写出。SRC 不能用立即数。

②.除法指令对所有条件码均无定义。“%”为取余运算符。

③.除法指令中如除数过小，则会使商产生溢出，从而产生除数为 0 的 0 型中断。

5. 十进制调整指令

(0).BCD 码概述

1).BCD 码(Binary Coded Decimal)：用 4 位二进制数表示一位十进制数的编码方法。此处的 BCD 码当作无符号数计算(它可用一个单独的字节作为符号位)。

2).压缩 BCD 码(packed BCD format)：用一个字节 8 位的二进制数表示 2 个 BCD 码。又称为组合 BCD 码。

3).非压缩 BCD 码(unpacked BCD format)：用一个字节低 4 位表示 1 个 BCD 码，高 4 位没有意义。又叫扩展 BCD 码。因此数字的 ASCII 码是一种非压缩 BCD 码。

(1). 压缩的 BCD 码调整指令

1). DAA——加法的十进制调整指令：紧跟在 ADD 或 ADC 指令之后进行调整。

指令格式：DAA ；若((((AL)AND 0FH)>9)OR((AF)=1))则(AL)←(AL)+6，(AF)←1；

若(((AL)>9FH)OR((CF)=1))则(AL)←(AL)+60H，(CF)←1

2). DAS——减法的十进制调整指令：紧跟在 SUB 或 SBB 指令之后进行调整。

指令格式：DAS ；若((((AL)AND 0FH)>9)OR((AF)=1))则(AL)←(AL)–6，(AF)←1；

若(((AL)>9FH)OR((CF)=1))则(AL)←(AL)–60H，(CF)←1

说 明：①.参加加、减运算的两个数一定为压缩 BCD 码，且加减法指令的目的操作数为 AL。

②.除对 OF 标志位无定义外，影响其它所有条件码。

(2). 非压缩的 BCD 码调整指令

1). AAA——加法的 ASCII 调整指令：紧跟在 ADD 或 ADC 指令之后进行调整。

指令格式：AAA ；若((((AL)AND 0FH)>9)OR((AF)=1))则(AL)←(AL)+6，(AH)←(AH)+1，(AF)←1，(CF)←(AF)，(AL)←((AL)AND 0FH)；否则(AL)←((AL)AND 0FH)

2). AAS——减法的 ASCII 调整指令：紧跟在 SUB 或 SBB 指令之后进行调整。

指令格式：AAS ；若((((AL)AND 0FH)>9)OR((AF)=1))则(AL)←(AL)–6，(AH)←(AH)–1，(AF)←1，(CF)←(AF)，(AL)←((AL)AND 0FH)；否则(AL)←((AL)AND 0FH)

说 明：①.参加加、减运算的两个数一定为非压缩的 BCD 码，且加减法指令的目的操作数为 AL。

②.影响 AF、CF，其余标志位无定义。

3). AAM——乘法的 ASCII 调整指令：紧跟在 MUL 指令之后进行调整。

指令格式：AAM ；(AH)←(AL)/0AH，(AL)←(AL)% 0AH

说 明：①.参加乘法运算的两数一定为高 4 位为 0 的非压缩 BCD 码，乘积在 AL 寄存器中。

②.影响 SF、ZF、PF，其余标志位无定义。

③.调整方法为：把 AL 寄存器的内容除以 0AH，商放在 AH 中，余数放在 AL 中。

4). AAD——除法的 ASCII 调整指令：在 DIV 指令之前调整。在 DIV 指令之后再用 AAM 调整。

指令格式：AAD ；(AL)←10×(AH)+(AL)，(AH)←0

说 明：①.被除数是存放在 AX 中的两位高 4 位为 0 的非压缩 BCD 码，除数也为高 4 位为 0 的非压缩 BCD 码。

②.影响 SF、ZF、PF，其余标志位无定义。

③.调整方法为：在 DIV 指令之前把两位高 4 位为 0 的非压缩 BCD 码的被除数调整为二进制数，再运行 DIV 指令，在 DIV 指令之后再用 AAM 指令将商调整为高 4 位为 0 的非压缩 BCD 码。

④.AAD 应用举例：求 73÷2=?

```
MOV   AX, 0703H      ；(AH)=07H，(AL)=03H。(即 73 的非组合 BCD 码)
MOV   BL, 02H        ；(BL)=02H。(即 2 的非组合 BCD 码)
AAD                  ；(AL)=49H。(即 73)
DIV   BL             ；除法运算，(AL)=24H(商)，(AH)=01H(余数)
AAM                  ；(AH)=03H，(AL)=06H。(即商 36 的非组合 BCD 码)
```

3.3.3 逻辑指令

1. 逻辑运算指令：可以对双字、字或字节执行按位的逻辑运算。

(1). AND——逻辑与指令

指令格式：AND DST，SRC ；(DST)←(DST)∧(SRC)

(2). OR——逻辑或指令

指令格式：OR DST，SRC ；(DST)←(DST)∨(SRC)

(3). NOT——逻辑非指令

指令格式：NOT OPR ；(OPR)←($\overline{OPR}$)

(4). XOR——逻辑异或指令

指令格式：XOR DST，SRC ；(DST)←(DST)⊕(SRC)

(5). TEST——测试指令

指令格式：TEST OPR1，OPR2 ；(OPR1)∧(OPR2)

说 明：①.DST、OPR、OPR1 不允许使用立即数寻址方式。

②.DST 与 SRC 及 OPR1 与 OPR2 的双操作数指令不能同时是存储器操作数。

③.NOT 指令不影响标志位。其它四条指令使 CF=OF=0，AF 无定义，SF、ZF、PF 则根据运算结果设置。

2. 位测试并修改指令：386 及其后继机型增加了本组指令。

(1). BT——位测试指令

指令格式：BT DST，SRC ；把 DST 中由 SRC 所指定的位的值送往标志位 CF

(2). BTS——位测试并置 1 指令

指令格式：BTS DST，SRC ；把 DST 中由 SRC 所指定的位的值送往标志位 CF，并将 DST 中的该位置 1

(3). BTR——位测试并置 0 指令

指令格式：BTR DST，SRC ；把 DST 中由 SRC 所指定的位的值送往标志位 CF，并将 DST 中的该位置 0

(4). BTC——位测试并变反指令

指令格式：BTC DST，SRC ；把 DST 中由 SRC 所指定的位的值送往标志位 CF，并将 DST 中的该位变反

说 明：①.DST 可用除立即数外的任一种寻址方式指定一个字或双字。

②.SRC 可以使用字或双字的寄存器方式，也可用 8 位立即数方式，指定所要测试的位的位置(该数值应在 0~31 之间)。

③.本组指令影响 CF 位。其它标志位则无定义。

3. 位扫描指令：386 及其后继机型增加了本组指令。

(1). BSF——正向位扫描指令

指令格式：BSF REG，SRC ；指令从位 0 开始自右向左扫描源操作数，目的是检索第一个为 1 的位。如遇到第一个为 1 的位则将 ZF 位置 0，并把该位的位置装入目的寄存器中；如源操作数为 0，则将 ZF 位置 1，目的寄存器无定义。

(2). BSR——反向位扫描指令

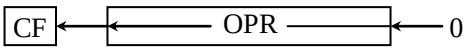
指令格式：BSR REG，SRC ；指令从最高有效位开始自左向右扫描源操作数，目的是检索第一个为 1 的位。该指令除方向与 BSF 相反外，其他同 BSF 指令。

说　　明：①.目的操作数必须用字或双字寄存器。
②.源操作数可以用除立即数外的任一种寻址方式指定一个字或双字。
③.本组指令影响 ZF 位。其它标志位则无定义。

4. 移位指令

(1). 移位指令

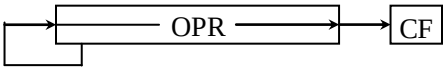
1). SHL——逻辑左移指令
指令格式：SHL　OPR，CNT　　；



2). SAL——算术左移指令
指令格式：SAL　OPR，CNT　　；同上

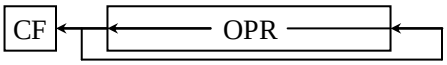
3). SHR——逻辑右移指令
指令格式：SHR　OPR，CNT　　；

4). SAR——算术右移指令
指令格式：SAR　OPR，CNT　　；

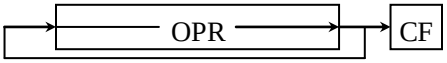


(2). 循环移位指令

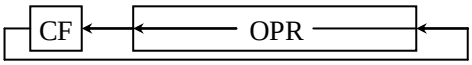
1). ROL——循环左移指令
指令格式：ROL　OPR，CNT　　；



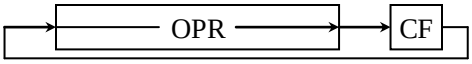
2). ROR——循环右移指令
指令格式：ROR　OPR，CNT　　；



3). RCL——带进位位循环左移指令
指令格式：RCL　OPR，CNT　　；



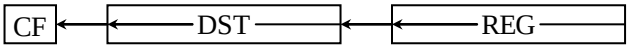
4). RCR——带进位位循环右移指令
指令格式：RCR　OPR，CNT　　；



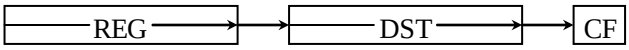
说　　明：①.OPR 为除立即数以外的任意寻址方式。移位次数由 CNT 决定，CNT=1 只移位 1 次；若移位次数超过 1 次，在 8086 中则 CNT 必须用 CL 代替。而在其他机型中也可用 8 位立即数指定范围从 1~31 的移位次数。
②.CF 位已在指令中给出其影响情况。OF 位只有在 CNT=1 时有效，一次移位前后的最高有效位(符号位)发生变化则 OF=1，否则 OF=0。循环指令不影响其它条件。移位指令由结果影响 SF、ZF、PF，而 AF 无定义。

(3). 双精度移位指令：386 及其后继机型增加了本组指令。

1). SHLD——双精度左移指令
指令格式：SHLD　DST，REG，CNT　；



2). SHRD——双精度右移指令
指令格式：SHRD　DST，REG，CNT　；



说　　明：①.DST 为除立即数以外的任意寻址方式指定字或双字操作数。
②.源操作数则只能使用寄存器方式指定与目的操作数相同长度的字或双字。
③.第三个操作数 CNT 用来指定移位次数，它可以是一个 8 位的立即数，也可以是 CL，用其内容存放移位计数值。数值范围应为 1~31，对于大于 31 的数，机器自动取模 32 的值来取代。
④.指令执行后，REG 不变，只取 DST 作为移位的结果。

3.3.4 串处理指令

用一条指令实现对一串字符或数据的操作。

1. 与 REP 相配合工作的 MOVS、STOS、LODS、INS 和 OUTS 指令

(1). REP 重复串操作直到计数寄存器 Count Reg (CX 或 ECX)的内容为 0 为止
指令格式：REP　string primitive　；其中 string primitive 可为 MOVS, STOS, LODS, INS 或 OUTS
执行操作：①.如(Count Reg)=0，则退出 REP，否则往下执行；
②.(Count Reg)←(Count Reg)−1
③.执行其后的串指令
④.重复①～③

(2). MOVS——串传送指令
指令格式：[REP]　MOVS　[ES:]DST，[Sreg:]SRC　；[Sreg:]为段跨越前缀。“[]”中为可选项。
[REP]　MOVSB　　；字节
[REP]　MOVSW　　；字

[REP] MOVSD ；双字(386 及其后继机型可用)

执行操作：((ES):(Destination-index))←((Sreg):(Source-index))。Sreg 缺省时为 DS。指针修改为：

字节操作：(Source-index)←(Source-index)±1，(Destination-index)←(Destination-index)±1

字 操 作：(Source-index)←(Source-index)±2，(Destination-index)←(Destination-index)±2

双字操作：(Source-index)←(Source-index)±4，(Destination-index)←(Destination-index)±4

说 明：①.在上述操作中，当方向标志 DF=0 时用“＋”，DF=1 时用“－”。可以使用指令 CLD 使 DF=0，STD 使 DF=1。

②.Source-index 为源变址寄存器，当其地址长度为 16 位时用 SI 寄存器，当其地址长度为 32 位时用 ESI 寄存器。

③.Destination-index 为目的变址寄存器，当其地址长度为 16 位时用 DI 寄存器，当其地址长度为 32 位时用 EDI 寄存器。以上 3 条说明适用于所有的串操作指令。

④.该指令不影响条件码。

补充说明：在执行串操作指令之前，应该先做好以下准备工作：

①.把存放在数据段中的源串首地址(如反向传送则为末地址)放入源变址寄存器中；

②.把将要存放数据串的附加段中的目的串首地址(或反向传送时的末地址)放入目的变址寄存器中；

③.把数据串长度放入计数寄存器(CX 或 ECX)；

④.建立方向标志。在完成这些准备工作后，就可以使用串指令传送信息了。

(3). STOS——存入串指令

指令格式：[REP] STOS [ES:]DST

[REP] STOSB ；字节

[REP] STOSW ；字

[REP] STOSD ；双字(386 及其后继机型可用)

字节操作：((Destination-index))←(AL)，(Destination-index)←(Destination-index)±1

字 操 作：((Destination-index))←(AX)，(Destination-index)←(Destination-index)±2

双字操作：((Destination-index))←(EAX)，(Destination-index)←(Destination-index)±4

(4). LODS——从串取指令

指令格式：LODS [Sreg:]SRC ；该指令与 REP 连用无多大意义

LODSB ；字节

LODSW ；字

LODSD ；双字

字节操作：(AL)←(Source-index)，(Source-index)←(Source-index)±1

字 操 作：(AX)←(Source-index)，(Source-index)←(Source-index)±2

双字操作：(EAX)←(Source-index)，(Source-index)←(Source-index)±4

(5). INS——串输入指令

指令格式：[REP] INS [ES:]DST，DX

[REP] INSB ；字节

[REP] INSW ；字

[REP] INSD ；双字(386 及其后继机型可用)

字节操作：((Destination-index))←((DX))(字节)，(Destination-index)←(Destination-index)±1

字 操 作：((Destination-index))←((DX))(字)，(Destination-index)←(Destination-index)±2

双字操作：((Destination-index))←((DX))(双字)，(Destination-index)←(Destination-index)±4

(6). OUTS——串输出指令

指令格式：[REP] OUTS DX，[Sreg:]SRC

[REP] OUTSB ；字节

[REP] OUTSW ；字

[REP] OUTSD ；双字(386 及其后继机型可用)

字节操作：((DX))←(Source-index)(字节)，(Source-index)←(Source-index)±1

字 操 作：((DX))←(Source-index)(字)，(Source-index)←(Source-index)±2

双字操作：((DX))←(Source-index)(双字)，(Source-index)←(Source-index)±4

2. 与 REPE/REPZ 和 REPNE/REPNZ 联合工作的 CMPS 和 SCAS 指令

(1). REPE/REPZ 当相等/为零时重复串操作

指令格式：REPE(或 REPZ) string primitive ；其中 string primitive 可为 CMPS 或 SCAS

执行操作：①.如(Count Reg)=0 或 ZF=0(即某次比较的结果两个操作数不等)时退出，否则往下执行；
②.(Count Reg)←(Count Reg)−1
③.执行其后的串指令
④.重复①～③

(2). REPNE/REPNZ 当不相等/不为零时重复串操作

指令格式：REPNE(或 REPNZ) string primitive ；其中 string primitive 可为 CMPS 或 SCAS
执行操作：①.如(Count Reg)=0 或 ZF=1(即某次比较的结果两个操作数相等)时退出，否则往下执行；
②.(Count Reg)←(Count Reg)−1
③.执行其后的串指令
④.重复①～③

(3). CMPS——串比较指令

指令格式：[REPE/REPNE] CMPS [Sreg:]SRC，[ES:]DST
[REPE/REPNE] CMPSB ；字节
[REPE/REPNE] CMPSW ；字
[REPE/REPNE] CMPSD ；双字(386 及其后继机型可用)
执行操作：((Sreg):(Source-index))–((ES):(Destination-index))。Sreg 缺省时为 DS。指针修改为：
字节操作：(Source-index)←(Source-index)±1，(Destination-index)←(Destination-index)±1
字 操 作：(Source-index)←(Source-index)±2，(Destination-index)←(Destination-index)±2
双字操作：(Source-index)←(Source-index)±4，(Destination-index)←(Destination-index)±4

(4). SCAS——串扫描指令

指令格式：[REPE/REPNE] SCAS [ES:]DST
[REPE/REPNE] SCASB ；字节
[REPE/REPNE] SCASW ；字
[REPE/REPNE] SCASD ；双字(386 及其后继机型可用)
字节操作：(AL)－(Destination-index)，(Destination-index)←(Destination-index)±1
字 操 作：(AX)－(Destination-index)，(Destination-index)←(Destination-index)±2
双字操作：(EAX)－(Destination-index)，(Destination-index)←(Destination-index)±4
说 明：①.串处理指令在不同的段之间传送或比较数据，如果需要在同一段内处理数据，可以在 DS 和 ES 中设置同样的地址，或者在源操作数字段使用段跨越前缀来实现。
②.当使用重复前缀时(Count Reg)是每次减 1 的，因此对于字或双字指令来说，预置时设置的值应该是字或双字的个数而不是字节数。

3.3.5 控制转移指令

1. 无条件转移指令

JMP——跳转指令，无条件地转移到指令指定的地址去执行从该地址开始的指令。分为段内转移(在同一段的范围内进行转移)和段间转移(转到另一个段去执行程序)。

(1). 段内直接短转移

指令格式：JMP SHORT OPR
执行操作：(IP)←(IP)+8 位位移量
386 及其后继机型则为：(EIP)←(EIP)+8 位位移量
如操作数长度为 16 位，则还需(EIP)←(EIP) AND 0000FFFFH
说 明：转移的目标地址 OPR 可直接使用符号地址(一个短标号)，而在机器执行时则是当前的 IP 或 EIP 的值(即 JMP 指令的下一条指令的地址)与指令中指定的 8 位位移量之和。相对位移量为 8 位(只需在–128~+127 字节范围内转移)。(条件转移只能用此方式。)

(2). 段内直接近转移

指令格式：JMP NEAR PTR OPR
执行操作：(IP)←(IP)+16 位位移量
386 及其后继机型则为：(EIP)←(EIP)+32 位位移量
如操作数长度为 16 位，则还需(EIP)←(EIP) AND 0000FFFFH
说 明：指令中给出一个相对位移量(实际是一个近标号)，这样有效转移地址为 IP 或 EIP 的当前值再加上一个 16 位(或 32 位)的位移量。OPR 可直接使用符号地址。

(3). 段内间接近转移

指令格式：JMP WORD PTR OPR

执行操作：(IP)←(EA)

386 及其后继机型则为：(EIP)←(EA)

如操作数长度为 16 位，则(EIP)←(EIP) AND 0000FFFFH

说　　明：有效地址 EA 值由 OPR 的寻址方式确定。它可以使用除立即数方式以外的任一种寻址方式。如果指定的是寄存器，则把寄存器的内容送到 IP 或 EIP 寄存器中；如果指定的是存储器中的一个字或双字，则把该存储单元的内容送到 IP 或 EIP 寄存器中。

(4). 段间直接远转移

指令格式：JMP FAR PTR OPR

执行操作：(IP)←OPR 的段内偏移地址，(CS)←OPR 所在段的段地址

386 及其后继机型则为：(EIP)←OPR 的段内偏移地址，(CS)←OPR 所在段的段地址

如操作数长度为 16 位，则还需(EIP)←(EIP) AND 0000FFFFH

说　　明：指令的操作数是一个远标号，该标号在另一个代码段内。指令的操作是将标号的偏移地址送 IP 或 EIP，段地址送 CS。

(5). 段间间接转移：指令的操作数为一个 32 位的存储器地址。指令的操作是将存储器的前两个字节送 IP，后两个字节送 CS，以实现到另一个代码段的转移。

指令格式：JMP DWORD PTR OPR

执行操作：(IP)←(EA)，(CS)←(EA+2)

386 及其后继机型则为：(EIP)←(EA)，(CS)←(EA+4)

如操作数长度为 16 位，则(EIP)←(EIP) AND 0000FFFFH

说　　明：有效地址 EA 值由 OPR 的寻址方式确定。它只能使用任一种存储器寻址方式。根据寻址方式求出 EA 后，把指定存储单元的字内容送到 IP 或双字内容送到 EIP 寄存器，并把下一个字的内容送到 CS 寄存器，这样就实现了段间跳转。

2. 条件转移指令

以某一个标志位的值或者某几个标志位的值作为判断是否转移的依据。如果满足指令中所要求的条件，则产生转移；否则往下执行排在条件转移指令后面的一条指令。只能使用段内直接短转移的寻址方式。在 386 及其后继机型中，还提供了近转移格式，这样它就可以转移到段内的任何位置。

指令格式：Jcc short_label ；cc 表示条件，具体分为如下 4 组：

(1). 根据单个条件标志的设置情况转移。这组包括 10 种指令。它们一般适用于测试某一次运算的结果并根据其不同特征产生程序分支作不同处理的情况。

1). JZ (或 JE)——结果为零(或相等)则转移

指令格式：JZ (或 JE) OPR

测试条件：ZF=1

2). JNZ (或 JNE)——结果不为零(或不相等)则转移

指令格式：JNZ (或 JNE) OPR

测试条件：ZF=0

3). JS——结果为负则转移

指令格式：JS OPR

测试条件：SF=1

4). JNS——结果不为负则转移

指令格式：JNS OPR

测试条件：SF=0

5). JO——结果溢出则转移

指令格式：JO OPR

测试条件：OF=1

6). JNO——结果不溢出则转移

指令格式：JNO OPR

测试条件：OF=0

7). JP (或 JPE)——奇偶位为 1(偶数个 1)则转移

指令格式：JP (或 JPE) OPR

测试条件：PF=1

8). JNP (或 JPO)——奇偶位为 0(奇数个 1)则转移

指令格式：JNP (或 JPO) OPR
测试条件：PF=0

9). JB (或 JNAE，或 JC)——低于，或者不高于或等于，或进位为 1 则转移
指令格式：JB (或 JNAE，或 JC) OPR
测试条件：CF=1

10). JNB (或 JAE，或 JNC)——不低于，或者高于或等于，或进位为 0 则转移
指令格式：JNB (或 JAE，或 JNC) OPR
测试条件：CF=0

(2). 比较两个无符号数，并根据比较的结果转移

1). JB (或 JNAE，或 JC)——低于，或者不高于或等于，或进位为 1 则转移(即“＜”)
指令格式：JB (或 JNAE，或 JC) OPR
测试条件：CF=1

2). JNB (或 JAE，或 JNC)——不低于，或者高于或等于，或进位为 0 则转移(即“≥”)
指令格式：JNB (或 JAE，或 JNC) OPR
测试条件：CF=0

3). JBE (或 JNA)——低于或等于，或不高于则转移(即“≤”)
指令格式：JBE (或 JNA) OPR
测试条件：CF∨ZF=1

4). JNBE (或 JA)——不低于或等于，或者高于则转移(即“＞”)
指令格式：JNBE (或 JA) OPR
测试条件：CF∨ZF=0

(3). 比较两个带符号数，并根据比较的结果转移

1). JL (或 JNGE)——小于，或者不大于或等于则转移(即“＜”)
指令格式：JL (或 JNGE) OPR
测试条件：SF⊕OF=1

2). JNL (或 JGE)——不小于，或者大于或等于则转移(即“≥”)
指令格式：JNL (或 JGE) OPR
测试条件：SF⊕OF=0

3). JLE (或 JNG)——小于或等于，或不大于则转移(即“≤”)
指令格式：JLE (或 JNG) OPR
测试条件：(SF⊕OF)∨ZF=1

4). JNLE (或 JG)——不小于或等于，或者大于则转移(即“＞”)
指令格式：JNLE (或 JG) OPR
测试条件：(SF⊕OF)∨ZF=0

(4). 测试 CX 或 ECX 的值为 0 则转移指令

1). JCXZ——CX 寄存器的内容为 0 则转移
指令格式：JCXZ OPR
测试条件：(CX)=0

2). JECXZ——ECX 寄存器的内容为 0 则转移
指令格式：JECXZ OPR
测试条件：(ECX)=0

3. 条件设置指令

386 及其后继机型才提供的一组指令。条件转移指令是根据上一条刚执行完的指令所设置的条件码情况，来判断是否产生程序分支的。有时并不希望在这一点就产生程序分支，而是希望在其后运行的程序的另一个位置，根据这一点条件码设置来产生程序分支，这样就要求把这一点的条件码情况保存下来，以便在其后使用，条件设置指令就是根据这样的要求建立的。

指令格式：SETcc DST ；cc 表示条件

执行操作：DST 可使用寄存器或任一种存储器寻址方式，但只能指定一个字节单元。指令根据所指定的条件码情况，如满足条件则把目的字节置为 1；如不满足条件则把目的字节置为 0。指令本身不影响标志位。具体分为如下 3 组：

(1). 根据单个条件标志的的值把目的字节置为 1

1). SETZ (或 SETE)——结果为零(或相等)则目的字节置为 1
指令格式：SETZ (或 SETE) DST

测试条件：ZF=1

2). SETNZ (或 SETNE)——结果不为零(或不相等)则目的字节置为 1
指令格式：SETNZ (或 SETNE) DST
测试条件：ZF=0

3). SETS——结果为负则目的字节置为 1
指令格式：SETS DST
测试条件：SF=1

4). SETNS——结果不为负则目的字节置为 1
指令格式：SETNS DST
测试条件：SF=0

5). SETO——结果溢出则目的字节置为 1
指令格式：SETO DST
测试条件：OF=1

6). SETNO——结果不溢出则目的字节置为 1
指令格式：SETNO DST
测试条件：OF=0

7). SETP (或 SETPE)——奇偶位为 1(偶数个 1)则目的字节置为 1
指令格式：SETP (或 SETPE) DST
测试条件：PF=1

8). SETNP (或 SETPO)——奇偶位为 0(奇数个 1)则目的字节置为 1
指令格式：SETNP (或 SETPO) DST
测试条件：PF=0

9). SETC (或 SETB，或 SETNAE)——进位为 1，或低于，或者不高于或等于则目的字节置为 1
指令格式：SETC (或 SETB，或 SETNAE) DST
测试条件：CF=1

10). SETNC (或 SETNB，或 SETAE)——进位为 0，或不低于，或者高于或等于则目的字节置 1
指令格式：SETNC (或 SETNB，或 SETAE) DST
测试条件：CF=0

(2). 比较两个无符号数，并根据比较的结果把目的字节置为 1

1). SETB (或 SETNAE，或 SETC)——低于，或不高于或等于，或进位为 1 则目的字节置 1(即“＜”)
指令格式：SETB (或 SETNAE，或 SETC) DST
测试条件：CF=1

2). SETNB (或 SETAE，或 SETNC)——不低于,或高于或等于，或进位为 0 则目的字节置 1(即“≥”)
指令格式：SETNB (或 SETAE，或 SETNC) DST
测试条件：CF=0

3). SETBE (或 SETNA)——低于或等于，或不高于则目的字节置为 1(即“≤”)
指令格式：SETBE (或 SETNA) DST
测试条件：CF∨ZF=1

4). SETNBE (或 SETA)——不低于或等于，或者高于则目的字节置为 1(即“＞”)
指令格式：SETNBE (或 SETA) DST
测试条件：CF∨ZF=0

(3). 比较两个带符号数，并根据比较的结果把目的字节置为 1

1). SETL (或 SETNGE)——小于，或者不大于或等于则目的字节置为 1(即“＜”)
指令格式：SETL (或 SETNGE) DST
测试条件：SF⊕OF=1

2). SETNL (或 SETGE)——不小于，或者大于或等于则目的字节置为 1(即“≥”)
指令格式：SETNL (或 SETGE) DST
测试条件：SF⊕OF=0

3). SETLE (或 SETNG)——小于或等于，或不大于则目的字节置为 1(即“≤”)
指令格式：SETLE (或 SETNG) DST
测试条件：(SF⊕OF)∨ZF=1

4). SETNLE (或 SETG)——不小于或等于，或者大于则目的字节置为 1(即“＞”)
指令格式：SETNLE (或 SETG) DST

测试条件：(SF⊕OF)∨ZF=0

4. 循环指令

(1). LOOP——循环指令

指令格式：LOOP OPR

测试条件：(Count Reg)≠0

(2). LOOPZ/LOOPE——当为 0 或相等时循环指令

指令格式：LOOPZ (或 LOOPE) OPR

测试条件：(Count Reg)≠0 且(ZF)=1

(3). LOOPNZ/LOOPNE——当不为 0 或不相等时循环指令

指令格式：LOOPNZ (或 LOOPNE) OPR

测试条件：(Count Reg)≠0 且(ZF)=0

说　　明：这三条指令的执行步骤是：

①.(Count Reg)←(Count Reg)–1

②.检查是否满足测试条件，如满足且操作数长度为 16 位，则(IP)←(IP)+D8 的符号扩展；如满足且操作数长度为 32 位，则(EIP)←(EIP)+D8 的符号扩展

5. 子程序(subroutine)

为便于模块化程序设计，往往把程序中某些具有独立功能的部分编写成独立的程序模块，称为子程序(也称为过程)。为此需要提供子程序(过程)的调用和返回指令。

(1). CALL——子程序(过程)调用指令

1). 段内直接近调用：指令操作数是一个近过程，该过程在本段内。

指令格式：CALL DST

执行操作：当操作数长度为 16 位时，Push (IP)
(IP)←(IP)+D16
或(EIP)←((EIP)+D16) AND 0000FFFFH
当操作数长度为 32 位时，Push (EIP)
(EIP)←(EIP)+D32

说　　明：指令中 DST 给出转向地址(即子程序的入口地址)，D16 即为机器指令中的位移量。

2). 段内间接近调用：指令的操作数是一个寄存器或存储器地址，其内容是一近过程的入口地址。

指令格式：CALL DST

执行操作：当操作数长度为 16 位时，Push (IP)
(IP)←(EA)
或(EIP)←(EA) AND 0000FFFFH
当操作数长度为 32 位时，Push (EIP)
(EIP)←(EA)

说　　明：指令中 DST 为除立即数以外的任一种寻址方式，由指定的寄存器或存储单元的内容给出转向地址。

3). 段间直接远调用：指令的操作数是一个远过程，该过程在另一个代码段内。

指令格式：CALL DST

执行操作：当操作数长度为 16 位时，Push (CS)
Push (IP)
(IP)←DST 指定的偏移地址
(CS)←DST 指定的段地址
当操作数长度为 32 位时，Push (CS)
Push (EIP)
(EIP)←DST 指定的偏移地址
(CS)←DST 指定的段地址

4). 段间间接远调用：指令的操作数是一个存储器地址。

指令格式：CALL DST

执行操作：当操作数长度为 16 位时，Push (CS)
Push (IP)
(IP)←(EA)
(CS)←(EA+2)
当操作数长度为 32 位时，Push (CS)

Push (EIP)
(EIP)←(EA)
(CS)←(EA+4)

说　　明：指令中 DST 为任一种存储器寻址方式，由指定的存储单元的内容给出转向地址。

(2). RET——子程序(过程)返回指令

RET 指令放在子程序的末尾，它使子程序在功能完成后返回调用程序继续执行，而返回地址是调用程序调用子程序(或称转子)时存放在堆栈中的，因此 RET 指令的操作是返回地址出栈送 IP 或 EIP 寄存器(段内或段间)和 CS 寄存器(段间)。

1). 段内近返回

指令格式：RET　　；DEBUG 反汇编为 RET，机器码为 C3H

执行操作：当操作数长度为 16 位时，(IP)←Pop ( )

386 及其后继机型：(EIP)←(EIP) AND 0000FFFFH

当操作数长度为 32 位时，(EIP)←Pop ( )

2). 段内带立即数近返回

指令格式：RET　EXP　　；DEBUG 反汇编为 RET　n，机器码为 C2 xxxxH

执行操作：在完成与 1)的 RET 完全相同的操作后，还需要修改堆栈指针：

(SP 或 ESP)←(SP 或 ESP)+D16

说　　明：其中 EXP 是一个表达式，根据它的值计算出来的常数成为机器指令中的位移量 D16。主程序通过压入堆栈操作将一定的参数或参数地址传递给子程序。子程序运行过程中，使用了这些参数或参数地址，子程序返回时没有必要再将这些参数或参数地址保留在堆栈中，因而，可以在返回指令后面加上参数 EXP，以腾出那些无用的参数或参数地址所占的单元。

3). 段间远返回

指令格式：RET　　；DEBUG 反汇编为 RETF，机器码为 CBH

执行操作：当操作数长度为 16 位时，(IP)←Pop ( )

386 及其后继机型：(EIP)←(EIP) AND 0000FFFFH

(CS)←Pop ( )

当操作数长度为 32 位时，(EIP)←Pop ( )

(CS)←Pop ( ) (32 位数出栈，高 16 位废除)

4). 段间带立即数远返回

指令格式：RET　EXP　　；DEBUG 反汇编为 RETF　n，机器码为 CA xxxxH

执行操作：在完成与 3)的 RET 完全相同的操作后，还需要修改堆栈指针：

(SP 或 ESP)←(SP 或 ESP)+D16

说　　明：①.从近过程返回和从远过程返回的指令是一样的，但机器编码不一样。段内返回指令的代码为 C3H，段间返回指令的代码为 CBH。

②.RET　EXP 指令为带参数的返回指令；EXP 可为 0~FFFFH 范围中的任何一个偶数。

6. 中断

(1). 中断的概念

1). 中断：当系统运行或者程序运行期间遇到某些特殊情况时，需要计算机自动执行一组专门的服务程序来进行处理，这种情况称为中断。中断分为内部中断和外部中断。

a. 内部中断：由中断指令或者是程序运行结果产生的中断称为内部中断。

b. 外部中断：由于外部 I/O 设备随机请求而产生的中断称为外部中断。

2). 中断服务程序：在中断中运行的一组服务程序称为中断服务程序或中断子程序。

3). 中断向量：中断服务程序的入口地址。

4). 中断类型码 N：为了区分各种中断而给每个中断按序从 0～FFH 编的号码称为中断类型码，用 N 表示。

5). 中断向量表：8086/8088 CPU 将所有的中断向量按中断类型码顺序存放在内存的起始 1KB 地址单元中，这 1KB 地址单元就称为中断向量表。

(2). INT——中断调用指令

指令格式：INT　TYPE

或 INT　　；TYPE=3 时，缺省

执行操作：Push (FLAGS)

IF←0
TF←0
AC←0
Push (CS)
Push (IP)
(IP)←(TYPE*4) (每个中断向量占 4 个字节)
(CS)←(TYPE*4+2)

说　　明：①.其中 TYPE 为类型号，它可以是常数或常数表达式，其值需在 0~255 范围内。
②.类型 0 的中断称为除数为 0 中断，由 CPU 自动产生，不能用中断指令调用。
③.类型 1 的中断称为单步中断，CPU 进入单步中断的依据为(TF)=1。不能用中断指令来调用。单步中断由调试程序 DEBUG 使用。
④.类型 2 的中断称为非屏蔽中断，属硬件中断，紧急情况使用，不许用中断指令来调用。
⑤.类型 3 的中断称为断点中断。用在调试程序中。INT 又称为断点中断指令，它是单字节指令。与其他 INT TYPE 不同，是双字节指令。
⑥.类型 4 的中断称为溢出中断。有专门的溢出中断调用指令 INTO。无 INT 4 指令。见下面。

(3). INTO——若溢出则中断指令

指令格式：INTO
执行操作：若 OF=1，则：Push (FLAGS)
IF←0
TF←0
AC←0
Push (CS)
Push (IP)
(IP)←(10H)
(CS)←(12H)

(4). IRET——从中断返回指令

指令格式：IRET ；适用于操作数长度为 16 位的情况
执行操作：(IP)←Pop ( )
(CS)←Pop ( )
(FLAGS)←Pop ( )

(5). IRETD——从中断返回指令

指令格式：IRETD ；适用于操作数长度为 32 位的情况
执行操作：(EIP)←Pop ( )
(CS)←Pop ( )
(EFLAGS)←Pop ( )

3.3.6 处理器控制与杂项操作指令

1. 标志处理指令

(1). CLC——清进位标志指令
指令格式：CLC ；(CF)←0

(2). CMC——对进位标志求反指令
指令格式：CMC ；(CF)←($\overline{CF}$)

(3). STC——置进位标志指令
指令格式：STC ；(CF)←1

(4). CLD——清方向标志指令
指令格式：CLD ；(DF)←0

(5). STD——置方向标志指令
指令格式：STD ；(DF)←1

(6). CLI——清中断允许标志指令
指令格式：CLI ；(IF)←0

(7). STI——置中断允许标志指令

指令格式：STI ；(IF)←1

(8). 设置单步标志程序段：置(TF)=1 程序。

```
PUSHF
POP    AX              ；(AX)←(FLAGS)
OR     AX，0100H           ；(TF)←1
PUSH   AX
POPF
```

(9). 清除单步标志程序段：清(TF)=0 程序。

```
PUSHF
POP    AX              ；(AX)←(FLAGS)
AND    AX，0FEFFH      ；(TF)←0
PUSH   AX
POPF
```

2. 其它处理器控制与杂项操作指令

(1). NOP(No operation)——空操作指令：CPU 执行指令时不进行任何操作，但占用 3 个时钟周期和一个字节的空间，然后继续执行下一条指令。起延时作用或为其他指令保留存储空间。

指令格式：NOP ；空操作，占用 3 个时钟周期

(2). HLT(Halt)——暂停指令：执行 HLT 指令后，CPU 进入暂停状态，CS 和 IP 指向 HLT 后面的一条指令的地址。外部中断或复位信号 RESET 可使 CPU 退出暂停状态。

指令格式：HLT ；暂停指令的执行

(3). ESC(Escape)——交权指令

指令格式：ESC ext_op，reg/mem；ext_op 是外操作码(协处理器的操作码)

(4). WAIT(Wait while TEST pin not asserted)——等待指令：8086 在执行 WAIT 指令的过程中，不断检测 TEST 引腿上的信号；而协处理器在完成工作以后，会往 8086 的 TEST 引腿送入一个低电平信号。8086 检测到此信号以后，便退出等待状态。

指令格式：WAIT ；等待协处理器操作结束

1). 外部中断可使 CPU 离开等待状态，但中断返回后又回到等待状态。

2). 一般在用 ESC 指令前先用一条 WAIT 指令，以防 8086 同时让协处理器干两件及两件以上的事。这是协处理器不允许的。

(5). LOCK(Lock bus)——总线封锁指令：指令前缀。

1). LOCK 指令前缀是一个特殊的可以放在任何指令前面的单字节指令前缀。

2). 该指令前缀迫使 CPU 封锁总线，并在 $\overline{\text{LOCK}}$ 线输出低电平，直到执行完前缀后面的指令为止。外部硬件可接收这个 $\overline{\text{LOCK}}$ 信号，而无法得到总线控制权。

(6). BOUND——界限指令(286 及其后继机型可用)

指令格式：BOUND reg，mem

执行操作：BOUND 指令检查给出的数组下标是否在规定的上下界之内。在则执行下一条指令；如超出了上下界范围，则产生中断 5。如发生中断，则返回时返回地址仍指向 BOUND 指令，而不是其下一条指令。

(7). ENTER——建立堆栈帧指令(286 及其后继机型可用，堆栈帧在第 6 章介绍)

指令格式：ENTER imm16，imm8

执行操作：指令中所给出的两个操作数均为立即数。第一个操作数为 16 位立即数，由其指定堆栈帧的大小，即其所占据的字节数。第二个操作数为 8 位立即数，它给出过程的嵌套层数，此数的范围应为 0~31。该指令完成以下操作：

①.Push (BP)或 Push (EBP)，以保存该寄存器的原始内容。

②.(BP)←(SP)或(EBP)←(ESP)，使 BP 或 EBP 寄存器保存当前堆栈指针 SP 或 ESP 的内容，以便在过程运行期间，以 BP(或 EBP)为基准访问堆栈帧中存放的变量。

③.(SP)←(SP)－imm16 或(ESP)←(ESP)－imm16，这样就建立了堆栈帧所占有的存储空间。

(8). LEAVE——释放堆栈帧指令(286 及其后继机型可用)

指令格式：LEAVE

执行操作：该指令在程序中位于退出过程的 RET 指令之前，用来释放由 ENTER 指令建立的堆栈帧存储区。该指令完成以下操作：

①.(SP)←(BP)或(SP)←(EBP)

②.(BP)←Pop ( )或(EBP)←Pop ( )

第 4 章 汇编语言程序格式

【教学目的】

本章内容也是本课程的重点，通过本章学习，使学生明确汇编语言的程序格式及程序设计方法，掌握汇编程序 MASM、连接程序 LINK 及调试程序 DEBUG 等的功能和用法，掌握 MASM 和 LINK 所用到的伪操作。特别是汇编语言程序的上机操作方法，为该课程实验打下基础。

【重点难点】

段定义伪操作、数据定义伪操作等的格式和用法，汇编语言程序格式、表达式和运算符，MASM、LINK 及 DEBUG 等的功能和用法。

【课 时 数】

7 学时。

4.1 汇编程序功能

1. 汇编程序(MASM)：把用户编写的汇编语言源程序翻译成机器语言目标程序的一种系统程序。
2. 汇编语言源程序：用汇编语言编写的程序称为汇编语言的源程序(扩展名为“.ASM”)。
3. 汇编程序的作用：把汇编语言源程序转换成用二进制代码表示的目标文件(称为“.OBJ”文件)。
4. 在计算机上运行汇编语言程序的步骤是
(1). 用编辑程序建立 ASM 源文件；
(2). 用 MASM 程序把 ASM 文件转换成 OBJ 文件；
(3). 用 LINK 程序把 OBJ 文件转换成 EXE 文件；
(4). 用 DOS 命令直接键入文件名就可执行该程序。
5. 汇编程序的主要功能
(1). 检查汇编语言源程序。
(2). 测出源程序中的语法错误，并给出出错信息。
(3). 产生源程序的目标程序，并给出列表文件。
(4). 展开宏指令。

4.2 伪操作

伪操作(伪指令)：用来为汇编程序提供某些信息，让汇编程序在汇编过程中执行某些特定功能的指令叫伪指令。它不是 CPU 指令系统中的指令。要注意的是：指令是指挥 CPU 执行什么操作，而伪操作是指挥 MASM 怎样工作。

4.2.1 处理器选择伪操作

这一组伪操作的功能是要告诉汇编程序应该选择哪一种指令系统。主要有以下几种：

```
.8086     ；选择 8086 指令系统，缺省时的默认值即为此
.286      ；选择 80286 指令系统
.286P     ；选择保护方式下的 80286 指令系统
.386      ；选择 80386 指令系统
.386P     ；选择保护方式下的 80386 指令系统
.486      ；选择 80486 指令系统
.486P     ；选择保护方式下的 80486 指令系统
.586      ；选择 Pentium 指令系统
.586P     ；选择保护方式下的 Pentium 指令系统
```

这类伪操作一般放在整个程序的最前面，也可放在程序中间。

4.2.2 段定义伪操作

1. 完整的段定义伪操作

(1). SEGMENT/ENDS——段定义伪操作：此对伪操作可以将汇编语言源程序分成几个段，通常为数据段、堆栈段、附加段和代码段。

伪操作格式：segname SEGMENT [align_type][combine_type][use_type] [‘class’]

⋮

segname ENDS

说　　明：①.定位类型(align_type)：说明段的起始地址应有怎样的边界值。它们可以是：

PARA：指定段的起始地址必须从小段边界开始，即段地址必须能被 16 整除。这样起始偏移地址可以从 0 开始。缺省时默认为 PARA。

BYTE：该段可以从任意地址开始。这样起始偏移地址可能不是 0。

WORD：该段必须从字的边界开始，即段地址必须为偶数。

DWORD：该段必须从双字的边界开始，即段地址必须能被 4 整除。

PAGE：该段必须从页的边界开始，即段地址必须能被 256 整除。

②.组合类型(combine_type)：说明程序连接时的段合并方法。它们可以是：

PRIVATE：该段为私有段(默认)，在连接时将不与其它模块中的同名分段合并。

PUBLIC：该段连接时将与有相同名字的其它分段连接在一起。连接次序由 LINK 指定。每一分段都从小段的边界开始，因此原有段之间有空隙。

COMMON：该段连接时与其它同名分段有相同的起始地址，所以会产生覆盖。COMMON 连接后的长度是各分段中长度最长的那个段的长度。

AT expression：使段起始地址是表达式计算出来的 16 位值。不能指定代码段。

MEMORY：与 PUBLIC 同义。

STACK：把不同模块中的同名段组合而形成一个堆栈段。其长度为原有段的长度总和。栈顶可自动指向连接后形成的大堆栈段的栈顶。

③.使用类型(use_type)：只适用于 386 及其后继机型，它用来说明使用 16 位寻址方式还是 32 位寻址方式。它们可以是：

USE16：使用 16 位寻址方式(默认)。段长不超过 64KB，地址形式为 16 位段地址和 16 位偏移地址。

USE32：使用 32 位寻址方式。段长可达 4GB，地址形式为 16 位段地址和 32 位偏移地址。

④.类别(‘class’)：在引号中给出连接时用于组成段组的类型名。类别说明并不能把相同类别的段合并起来，但在连接后形成的装入模块中，可以把它们的位置靠在一起。

(2). ASSUME——段指定伪操作：告诉汇编程序，段和段寄存器的对应关系。

伪操作格式：ASSUME 分配(assignment)，…，assignment

说　　明：其中 assignment 说明分配情况，其格式为：

段寄存器名(segment register name)：段名字(segment name)[，段寄存器名：段名字[，……]]

ASSUME NOTHING 则可取消前面由 ASSUME 所指定的段寄存器。

2. 存储模型与简化段定义伪操作

(1). MODEL——存储模型(memory_model)伪操作，即用来说明在存储器中是如何安放各个段的。

伪操作格式：. MODEL memory_model [，model options]

说　　明：根据它们的不同组合，可以建立如下七种存储模型：

①.Tiny 所有数据和代码都放在一个段内，其数据和代码都是近访问。

②.Small 所有数据放在一个 64KB 的数据段内，所有代码放在另一个 64KB 的代码段内，数据和代码也都是近访问的。这是一般最常用的一种模型。

③.Medium 代码使用多个段，一般一个模块一个段，而数据则合并成一个 64KB 的段组。这样，数据是近访问的，而代码则可远访问。

④.Compact 所有代码都放在另一个 64KB 的代码段内，数据则可放在多个段内，形成代码是近访问的，而数据则可远访问的格式。

⑤.Large 代码和数据都可用多个段，所以数据和代码都可以远访问。

⑥.Huge 与 Large 模型相同，其差别是允许数据段的大小超过 64KB。

⑦.Flat 允许用户用 32 位偏移量，但 DOS 下不允许使用这种模型，只能在 OS/2 下或其他保护模式的操作系统下使用。MASM5 不支持，但 MASM6 支持。

model options 允许用户指定三种选项：高级语言接口、操作系统和堆栈距离。

①.高级语言接口　是指该汇编语言程序作为某一种高级语言程序的过程而为该高级语言程序调用时，应该用如 C, BASIC，FORTRAN，PASCAL 等来加以说明。

②.操作系统　是要说明程序运行于哪个操作系统之下，可用 OS_DOS 或 OS_OS2 来说明，默认项是 OS_DOS。

③.堆栈距离　可用 NEARSTACK 或 FARSTACK 来说明。其中 NEARSTACK 是指把堆栈段和数据段组合到一个 DGROUP 段中，DS 和 SS 均指向 DGROUP 段；FARSTACK 是指堆栈段和数据段并不合并。

(2). 简化的段定义伪操作

1). 汇编程序给出的标准段有下列几种：这种分段方法把数据段分得更细：一是把常数段和数据段分开；二是把初始化数据段和未初始化数据段分开(其中初始化数据段是指程序中已指定初始值的数据)；三是把近和远的数据段分开。这样做的结果可便于与高级语言兼容。

a. code	代码段
b. initialized data	初始化数据段
c. uninitialized data	未初始化数据段
d. far initialized data	远初始化数据段
e. far uninitialized data	远未初始化数据段
f. constants	常数段
g. stack	堆栈段

2). 对应以上的标准段，可有如下简化段伪操作

伪操作格式：**.** CODE [name]　；对于一个代码段的模型，段名为可选项；
　　对于多个代码段的模型，则应为每个代码段指定段名

.DATA

.DATA?

.FARDATA [name]　；可指定段名。如不指定，则将以 FAR_DATA 命名。

.FARDATA? [name]　；可指定段名。如不指定，则将以 FAR_BSS 命名。

.CONST

.STACK [size]　；可指定堆栈段大小。如不指定，则默认值为 1KB。

说　　明：当使用简化段伪操作时，必须在这些简化段伪操作出现之前，即程序的一开始先用**.**MODEL 伪操作定义存储模型，然后再用简化段伪操作定义段。每一个新段的开始就是上一段的结束，而不必用 ENDS 作为段的结束符。

(3). 与简化段定义有关的预定义符号：汇编程序给出了与简化段定义有关的一组预定义符号，它们可在程序中出现，并由汇编程序识别使用。如预定义符号@data 就给出了数据段的段名。另有一些预定义符号，也可与条件汇编伪操作相配合，以帮助用户编写一些较为复杂的代码。

(4). 用 MODEL 定义存储模型时的段默认属性：参见书 123 页表 4.1 所示。

(5). 简化段定义举例

3. GROUP——段组定义伪操作：在各种存储模型中，汇编程序自动地把各数据段组成一个段组 DGROUP，以便程序在访问各数据段时使用一个数据段寄存器 DS。GROUP 伪操作允许用户自行指定段组，其格式如下：

伪操作格式：grpname　GROUP　segname [，segname…]

说　　明：其中 grpname 为段组名，segname 则为段名。

4.2.3 程序开始和结束伪操作

1. NAME——指定模块名伪操作

伪操作格式：NAME　module_name　；汇编程序将以给出的模块名作为该模块的名字

2. TITLE——指定打印标题伪操作：指定每一页上的打印标题。同时，如果程序中没有 NAME 伪操作，则汇编程序将用 text 中的前 6 个字符作为模块名。text 最多可有 60 个字符。如果程序中既没有 NAME 又没有 TITLE 伪操作，则将用源文件名作为模块名。所以 NAME 和 TITLE 伪操作并非必须的。

伪操作格式：TITLE　text

3. END——源程序结束伪操作

伪操作格式：END　[label]　；其中 label 指示程序开始执行的起始地址

说　　明：如果多个程序模块相连接，则只有主程序要使用标号，其他子程序模块则只用 END 而不必指定标号。

4. MASM6.0 版的汇编程序还增加了定义程序的入口点和出口点的伪操作

伪操作格式：.STARTUP ；用来定义程序的初始入口点，并且产生设置 DS、SS 和 SP 的代码。

伪操作格式：.EXIT [return_value] ；用来产生退出程序并返回操作系统的代码，其中 return_value 为返回给操作系统的值。常用 0 作为返回值。

4.2.4 数据定义及存储器分配伪操作：此类伪操作的通用格式为：

[Variable] Mnemonic Operand [，Operand…] [；Comments]

1. 变量名(Variable)：可有可无，用符号地址表示。后面跟空格或 TAB 分隔符，不能跟“：”。其值等于第一个字节的偏移地址。

2. 注释(Comments)：可有可无，用于说明该伪操作的功能。

3. 助记符(Mnemonic)：说明所用伪操作的助记符名称同时也说明所定义的数据类型。通常为下面 6 种。

(1). DB(Define Byte)：定义字节伪操作，其后的每个操作数都占有一个字节(8 位)。

(2). DW(Define Word)：定义字伪操作，其后的每个操作数都占有一个字(16 位)。

(3). DD(Define Double Word)：定义双字伪操作，其后的每个操作数都占有一个双字(32 位)。

(4). DF(Define Far Pointer)：定义 6 个字节伪操作，用于存放远地址指针，其后的每个操作数都占有 6 个字节(48 位)。只能用于 386 及其后继机型。

(5). DQ(Define Quad Word)：定义 4 字伪操作，其后的每个操作数都占有 4 个字(64 位)。

(6). DT(Define Ten Bytes)：定义十字节伪操作，其后的每个操作数都占有十字节，形成压缩 BCD 码。

4. 操作数(Operand)：一个常数、一个其值为常数的表达式，或是一个字符串常数。可以是问号“?”及重复操作符 DUP 等。

(1).“表达式”作为操作数(常数是一种特殊的表达式)：可用定义的变量或标号组成表达式。

1). 变量±常数表达式

2). 标号±常数表达式

(2). 字符串作为操作数。

(3).“?” 操作数：仅给变量保留相应的存储空间，而不赋给变量初值。

(4). 重复操作符 DUP：用来指定某个(或某些)操作数重复的次数(还可以嵌套)。形式为：

repeat_count DUP (初值[，初值…]) ；repeat_count 为重复的次数

(5). PTR 属性操作符：指定操作数的类型属性为 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE 类型。

操作符格式：类型(Type) PTR 变量(Variable)±常数表达式(Constant expression)

(6). LABEL——伪操作：指定一个操作数具有不同的类型属性。

伪操作格式：name LABEL Type

说 明：①.对于数据项，name 用变量名(variable_name)表示。类型为 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE。

②.对于可执行的程序代码，name 用标号名(label_name)表示。类型为 NEAR、FAR。

4.2.5 表达式赋值伪操作 EQU、= ：(又称为符号定义伪操作、符号赋值伪操作。) 有时程序中多次出现同一个表达式，为方便起见可以用赋值伪操作给表达式赋予一个名字。

1. EQU——表达式赋值伪操作：用来对一个名字进行赋值。但不能对一个已赋值的名字重新赋值。

伪操作格式：表达式名字 EQU 表达式 ；表达式可以为常数或者数据的地址

说 明：PURGE 语句：用来解除对某一个名字的赋值。以后此名字才可由 EQU 重新赋值。

伪操作格式：PURGE 原名字 ；原名字，即前面已赋过值的名字

2. = ——表达式赋值伪操作：“=”(等号)伪操作的功能与 EQU 伪操作基本相同，主要区别在于它可以对同一个名字重新赋值。

伪操作格式：名字 = 表达式 ；表达式可以为常数或者数据的地址

4.2.6 地址计数器与对准伪操作

1. $——地址计数器：在汇编程序对源程序汇编的过程中，使用地址计数器(location counter)来保存当前正在汇编的指令的偏移地址。地址计数器的值可用$来表示，汇编语言允许用户直接用$来引用地址计数器的值。

(1). 当$用在指令中时，它表示本条指令的第一个字节的地址。

(2). 当$用在伪操作的参数字段时，则和它用在指令中的情况不同，它表示的是地址计数器的当前值。

2. ORG——地址对准伪操作：用来规定下一个字节的地址成为常数表达式的值。

伪操作格式：ORG 常数表达式(Constant expression)

3. EVEN——偶地址对准伪操作：用来规定下一个字节的地址成为偶数。

伪操作格式：EVEN

4. ALIGN——其它地址对准伪操作：使地址计数器成为 boundary 的整数倍，boundary 必须是 2 的幂。

伪操作格式：ALIGN boundary

4.2.7 基数控制伪操作

1. 汇编程序默认的数为十进制数。二进制数要以字母 B 结尾；八进制数要以字母 O 或 Q 结尾；十进制数要以字母 D 结尾或缺省；十六进制数要以字母 H 结尾，当首字符为 A～F 时前面必须加 0。

2. .RADIX——基数控制伪操作：可以把默认的基数改变为 2～16 范围内的任何基数。

伪操作格式：**.**RADIX 表达式(expression) ；表达式为十进制数，用来表示基数值

3. 字符串可以看成串常数，要用成对的单引号或双引号作为定界符，得到的是字符串的 ASCII 码值。

4.3 汇编语言程序格式

汇编语言源程序中的每个字句可以由 4 项组成。格式如下：

[名字(name)] 操作(operation) 操作数(operand) [；注释(comment)]

4.3.1 名字项(name)

1. 源程序中可用于名字的字符有：

(1). 字母 A～Z，a～z。

(2). 数字 0～9。数字不能用在名字的最前面。

(3). 专用字符？、 . 、@、_、$。如果用到“.”则必须为名字的第一个字符。

2. 名字项可以是标号或变量，用来表示本语句的符号地址。同一个名字不可重复定义。

(1). 标号：在代码段中定义，后面必须跟冒号“:”。标号有三种属性：段、偏移及类型。

1). 段属性：定义标号的段起始地址，此值必须在一个段寄存器中，而标号的段总是在 CS 寄存器中。

2). 偏移属性：标号的偏移地址是从段起始地址到定义标号的位置之间的字节数。

3). 类型属性：用来指出该标号是在本段内(NEAR)引用，还是在其它段(FAR)引用。

(2). 变量：在除代码段以外的其它段中定义，后面不能跟冒号。变量经常在操作数字段出现。变量也有三种属性：段、偏移及类型。

1). 段属性：定义变量的段起始地址，此值必须在一个段寄存器中。

2). 偏移属性：变量的偏移地址是从段起始地址到定义变量的位置之间的字节数。在当前段内给出变量的偏移值等于当前地址计数器的值，当前地址计数器的值可以用 **$** 来表示。

3). 类型属性：变量的类型属性定义该变量所保留的字节数。

4.3.2 操作项(operation)：指令、伪操作或宏指令的助记符。

4.3.3 操作数项(operand)：由一个或多个表达式组成，多个操作数项之间一般用逗号分开。操作数项可以是常数、寄存器、标号、变量或由表达式组成。表达式经汇编后有确定的结果，表达式是常数、寄存器、标号、变量与一些操作符相组合的序列，可以有数字表达式和地址表达式两种。用于组合表达式的操作符有：

1. 算术操作符：+、−、*、/、MOD。MOD 为取余运算。地址表达式只能用+、−、*、/。

2. 逻辑与移位操作符

(1). 逻辑操作符：AND、OR、XOR、NOT。逻辑运算为按位运算，只能用于数字表达式中。

(2). 移位操作符：汇编程序将 expression 左移或右移 numshift 位，如移位数大于 15，则结果为 0。

操作符格式：expression SHL (或 SHR) numshift

3. 关系操作符：EQ(相等)、NE(不等)、LT(小于)、GT(大于)、LE(小于或等于)、GE(大于或等于)6 种。关系操作符的两个操作数必须都是数字或是同一段内的两个存储器地址。计算结果为逻辑值，结果为真，表示为 0FFFFH；结果为假，表示为 0。

4. 数值回送(Value_returning)操作符：TYPE、LENGTH、SIZE、OFFSET、SEG 等。

(1). TYPE：如果是变量，则汇编程序将回送该变量的以字节数表示的类型：DB 为 1，DW 为 2，DD 为 4，DF 为 6，DQ 为 8，DT 为 10。如果是标号，则汇编程序将回送代表该标号类型数值：NEAR 为−1，FAR 为−2。如果表达式为常数，则应回送 0。

操作符格式：TYPE expression

(2). LENGTH：对于变量中使用 DUP 的情况，汇编程序将回送分配给该变量的单元数。否则回送 1。
操作符格式：LENGTH 变量(variable)

(3). SIZE：汇编程序将回送分配给该变量的字节数。但是此值是 LENGTH 值和 TYPE 值的乘积。
操作符格式：SIZE 变量(variable)

(4). OFFSET：汇编程序将回送变量或标号的偏移地址值。
操作符格式：OFFSET 变量(variable)或标号(label)

(5). SEG：汇编程序将回送变量或标号的段地址值。
操作符格式：SEG 变量(variable)或标号(label)

5. 属性操作符：PTR、段操作符、SHORT、THIS、HIGH、LOW、HIGHWORD 和 LOWWORD 等。

(1). PTR：可用以指定存储器操作数的类型。通常和伪操作 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE 等连起来使用。利用 PTR 运算符还可以建立一个新的存储器操作数，它与原来的同名操作数具有相同的段和偏移量，但可以有不同的类型。格式中的类型字段表示所赋予的新的类型属性，而表达式字段则是被取代类型的符号地址。
操作符格式：类型(type) PTR 表达式(expression)

(2). 段操作符：用来表示一个标号、变量或地址表达式的段属性。段跨越前缀即是其中一种。
操作符格式：段寄存器名(或段名或组名)：地址表达式

(3). SHORT：用来修饰 JMP 指令中转向地址的属性，指出转向地址是在下一条指令地址的±127 个字节范围内。

(4). THIS：可以象 PTR 一样建立一个指定类型(BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE)的或指定距离(NEAR、FAR)的地址操作数。该操作数的段地址和偏移地址与下一个存储单元地址相同。
操作符格式：THIS 属性(attribute)或类型(type)

(5). HIGH 和 LOW——字节分离操作符：它接收一个数或地址表达式，HIGH 取其高位字节，LOW 取其低位字节。

(6). HIGHWORD 和 LOWWORD——字分离操作符：它接收一个数或地址表达式，HIGHWORD 取其高位字，LOWWORD 取其低位字。

(7). 方括号“[ ]”操作符：经方括号“[ ]”括进去的内容表示存储器的地址。

6. 表达式中的多个运算符的运算规则

(1). 运算符的运算规则

1). 优先级高的先运算，优先级低的后运算。

2). 优先级相同时按表达式中从左到右的顺序运算。

3). 圆括号可提高运算符的优先级，圆括号内的运算总是在其任何相邻的运算之前进行。

(2). 各种运算符的优先级如下表：

优先级		运 算 符
高	1	LENGTH，SIZE，WIDTH，MASK，( )，[ ]，<>
	2	. (结构变量名后面的操作符)
	3	: (段跨越操作符)
	4	PTR，OFFSET，SEG，TYPE，THIS
	5	HIGH，LOW，HIGHWORD，LOWWORD
	6	+，–(一元运算符，即正、负)
	7	*，/，MOD，SHL，SHR
	8	+，–(二元运算符，即加、减)
	9	EQ，NE，LT，LE，GT，GE
	10	NOT
	11	AND
	12	OR，XOR
低	13	SHORT

4.3.4 注释项(comment)：用来说明一段程序或一条或几条指令的功能，必须用分号“;”开始，它是可有可无的。注释应该写出本条(或本段)指令在程序中的功能和作用，而不应该只写指令的动作。

汇 编 语 言 源 程 序 格 式 举 例(1)

```
; 这里是程序标题(PROGRAM TITLE GOES HERE——)
; 后面有说明语句(Followed by descriptive phrases)
; 这里是说明语句(EQU STATEMENTS GO HERE)
; ********************************************************
datarea  segment                        ; 定义数据段
; 这里是数据(DATA GO HERE)
datarea  ends
; ********************************************************
prognam  segment                        ; 定义代码段
; ------------------------------------------------------------------------
main     proc    far                    ; 主程序部分
         assume  cs: prognam, ds: datarea
start:                                  ; 开始执行的地址
; 设置返回到 DOS(set up stack for return)。如省略则后面的 ret 改用 int 20h 等即可。
         push    ds                     ; 保存老的数据段地址
         sub     ax, ax                 ; AX 清 0
         push    ax                     ; 此 3 条指令保护的是(ds):0000，即 PSP+0 的地址
; 设置 DS 指向当前数据段地址(set DS register to current data segment)
         mov     ax, datarea            ; 取数据段的段地址
         mov     ds, ax                 ; 保存到 DS 寄存器
; 这里是程序的主要部分(MAIN PART OF PROGRAM GOES HERE)
         ret                            ; 返回到 DOS
main     endp                           ; 主程序部分结束
; ------------------------------------------------------------------------
sub1     proc    near                   ; 定义子程序 1
; 这里是子程序部分(SUBROUTINE GOES HERE)
sub1     endp                           ; 子程序 1 结束
; ------------------------------------------------------------------------
prognam  ends                           ; 代码段结束
         end     start                  ; 源程序结束
```

汇 编 语 言 源 程 序 格 式 举 例(2)

```
         .model  small                  ; define memory model
         .stack  100h                     ; define stack segment
         .data                          ; define data segment
; 这里是数据(DATA GOES HERE)
         .code                          ; define code segment
main     proc    far                    ; 主程序部分
start:                                  ; 开始执行的地址
         mov     ax, @data                ; 取数据段的段地址
         mov     ds, ax                 ; 保存到 DS 寄存器
; 这里是程序的主要部分(MAIN PART OF PROGRAM GOES HERE)
         mov     ax, 4c00h
         int     21h                    ; 返回到 DOS
main     endp                           ; 主程序部分结束
         end     start                  ; 源程序结束
```

4.4 汇编语言程序的上机过程

4.4.1 建立汇编语言的工作环境：为运行汇编语言程序至少要在磁盘上建立以下文件：

(1). EDIT.EXE 全屏幕编辑程序，用于产生汇编语言源程序
(2). MASM.EXE 汇编程序
(3). LINK.EXE 连接程序
(4). DEBUG.EXE 调试程序

4.4.2 建立 ASM 文件：用全屏幕编辑程序建立汇编语言源程序(ex_movs.ASM)。

```
;PROGRAM TITLE GOES HERE——ex_movs.ASM
;*******************************************************
data    segment                         ;定义数据段
  source_buffer   db   40   dup('a')
data    ends
;*******************************************************
extra   segment                         ;定义附加数据段
  dest_buffer     db   40   dup(?)
extra   ends
;*******************************************************
code    segment                         ;定义代码段
;-------------------------------------------------------------------------
main    proc    far                     ;主程序部分
        assume  cs:code, ds:data, es:extra
start:                                  ;开始执行的地址
;设置返回到 DOS(set up stack for return)。如省略则后面的 ret 改用 int 20h 等即可。
        push    ds                      ;保存老的数据段地址
        sub     ax, ax                  ;AX 清 0
        push    ax                      ;此 3 条指令保护的是(ds):0000，即 PSP+0 的地址
;设置 DS 指向当前数据段地址(set DS register to current data segment)
        mov     ax, data                ;取数据段的段地址
        mov     ds, ax                  ;保存到 DS 寄存器
;设置 ES 指向当前数据段地址(set ES register to current data segment)
        mov     ax, extra               ;取附加数据段的段地址
        mov     es, ax                  ;保存到 ES 寄存器
;这里是程序的主要部分(MAIN PART OF PROGRAM GOES HERE)
        lea     si, source_buffer       ;put offset addr of source buffer in SI
        lea     di, dest_buffer         ;put offset addr of dest buffer in DI
        cld                             ;set DF flag to forward
        mov     cx, 40                  ;put count in CX
        rep     movsb                   ;move entire string
        ret                             ;返回到 DOS
main    endp                            ;主程序部分结束
;-------------------------------------------------------------------------
code    ends                            ;代码段结束
        end     start                   ;源程序结束
```

简化段定义的程序

```
;PROGRAM TITLE GOES HERE——ex_movs.ASM
;*******************************************************
.model  small                           ;定义存储类型
.stack  100h
.data                                   ;定义数据段
  source_buffer   db   40   dup('a')
;*******************************************************
```

```
.const                                          ;定义附加数据段，改用 FARDATA 也可
  dest_buffer       db    40    dup(?)
;*************************************************
.code                                           ;定义代码段
;-------------------------------------------------------------------------
main      proc      far                         ;主程序部分
start:                                          ;开始执行的地址
;设置 DS 指向当前数据段地址(set DS register to current data segment)
          mov       ax, @data                       ;取数据段的段地址
          mov       ds, ax                      ;保存到 DS 寄存器
;设置 ES 指向当前数据段地址(set ES register to current data segment)
          mov       ax, const                   ;取附加数据段的段地址，改用@FARDATA
          mov       es, ax                      ;保存到 ES 寄存器
          assume    es: const                   ;改用@FARDATA
;这里是程序的主要部分(MAIN PART OF PROGRAM GOES HERE)
          lea       si, source_buffer           ;put offset addr of source buffer in SI
          lea       di, dest_buffer             ;put offset addr of dest buffer in DI
          cld                                   ;set DF flag to forward
          mov       cx, 40                      ;put count in CX
          rep       movsb                       ;move entire string
          mov       ax, 4c00h                   ;返回到 DOS
          int       21h
main      endp                                  ;主程序部分结束
;-------------------------------------------------------------------------
          end       start                       ;源程序结束
```

4.4.3 用MASM程序产生OBJ 文件：将源程序(ex_movs.ASM)汇编成机器语言目标程序(ex_movs.OBJ)。

1. 操作命令

MASM ex_movs; ；最好在命令后加分号“;”，这样不产生其它文件，速度快。

2. 汇编程序还有一个重要功能：可以给出源程序中的错误信息。

(1). 警告错误(Warning Errors)：指出汇编程序所认为的一般性错误。(不影响汇编结果。)

(2). 严重错误(Severe Errors)：指出汇编程序认为已使汇编程序无法进行汇编的错误。并给出错误代号和信息。

4.4.4 用LINK程序产生EXE 文件：将目标程序(ex_movs.OBJ)连接成机器能执行的ex_movs.EXE 文件。

LINK ex_movs; ；最好在命令后加分号“;”这样只产生 EXE 文件，速度快。

4.4.5 程序的执行和调试

1. 在建立了 EXE 文件后，就可以直接从 DOS 执行程序，如下所示：

ex_movs↙

2. 用 DEBUG 调试程序，如下所示：

DEBUG ex_movs.exe↙

4.4.6 COM 文件

1. COM 文件也是一种可执行文件，由程序本身的二进制代码组成，没有 EXE 文件所具有的包括有关文件信息的标题区(header)，所以它占用的存储空间比 EXE 文件小。COM 文件不允许分段，它所占用的空间不许超过 64KB。因而只能用来编制较小的程序。

2. 用 EXE2BIN.EXE 程序将 filename.EXE 文件转换为 filename.COM 文件：

EXE2BIN filename filename.com

3. COM 文件还可以在调试程序 DEBUG 中用 A 或 E 命令建立。

第 5 章 循环与分支程序设计

【教学目的】

通过本章学习，使学生掌握程序设计的基本方法，并对两种基本的程序结构循环程序和分支程序的设计能够牢牢掌握。

【重点难点】

循环程序和分支程序的设计方法，编写汇编语言程序和上机操作。

【课 时 数】

4 学时。

1. 编制一个汇编语言程序的步骤如下：

(1). 分析题意，确定算法。

(2). 根据算法，画出程序框图。

(3). 根据框图编写程序。

(4). 上机调试程序。

2. 程序有四种结构形式：顺序、循环、分支、子程序。顺序程序结构是指完全按顺序逐条执行的指令序列。

5.1 循环程序设计

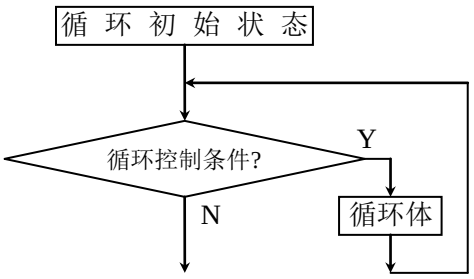
5.1.1 循环程序的结构形式

1. 循环程序的两大结构形式

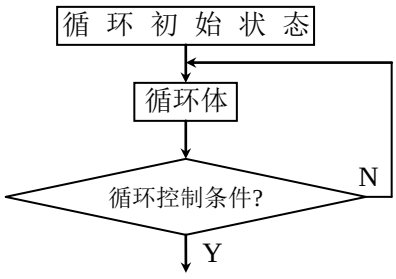
(1). DO_WHILE 结构：把对循环控制条件的判断放在循环的入口，先判断条件，满足条件就执行循环体，不满足就退出循环。

(2). DO_UNTIL 结构：先执行循环体然后再判断循环控制条件，不满足条件则继续执行循环操作，一旦满足条件就退出循环。

(3). 两种结构的框图形式表示



DO_WHILE 结构



DO_UNTIL 结构

2. 组成循环程序的三大部分

(1). 设置循环的初始状态。即循环初始化。

(2). 循环体。循环工作的主体，它由循环的工作部分和循环的修改部分组成。

1). 循环的工作部分：是为完成程序功能而设计的主要程序段；

2). 循环的修改部分：是为保证每一次循环时，参加执行的信息能发生有规律的变化而建立的程序段。

(3). 循环控制部分。每个循环程序必须选择一个循环控制条件来控制循环的运行和结束，而合理的选择该控制条件就成为循环程序设计的关键问题。

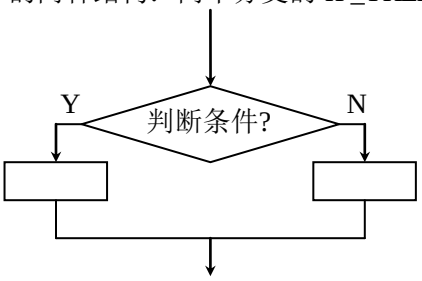
5.1.2 循环程序设计方法

5.1.3 多重循环程序设计：循环可以嵌套。多重循环程序设计的基本方法和单重循环程序设计是一致的，应分别考虑各重循环的控制条件及其程序实现，相互之间不能混淆。另外应该注意在每次通过外层循环再次进入内层循环时，初始条件必须重新设置。

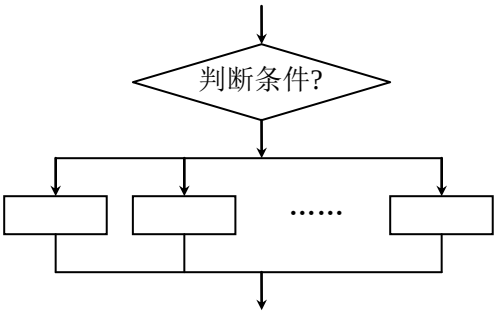
5.2 分支程序设计

5.2.1 分支程序的结构形式

1. 分支程序的两种结构：两个分支的 IF_THEN_ELSE 结构和多个分支的 CASE 结构。



IF_THEN_ELSE 结构



CASE 结构

2. 分支程序的特点：运行方向是向前的，在某一种确定条件下，只能执行多个分支中的一个分支。

5.2.2 分支程序设计方法：程序的分支一般用条件转移指令来实现。

5.2.3 跳跃表法：在实现 CASE 结构时还可以使用跳跃表法使程序能根据不同的条件转移到多个程序分支中去。

5.3 如何在实模式下发挥 80386 及其后继机型的优势

5.3.1 充分利用高档机的 32 位字长特性

5.3.2 通用寄存器可作为指针寄存器

5.3.3 与比例因子有关的寻址方式

5.3.4 各种机型提供的新指令

1. 286 提供的新指令

PUSHA / POPA
PUSH IMM
IMUL REG，SRC
IMUL REG，SRC，IMM
INS
OUTS
BOUND
ENTER
LEAVE

2. 386 提供的新指令

MOVSX，MOVZX
PUSHAD / POPAD
PUSHFD / POPFD
CDQ，CWDE
BT，BTS，BTR，BTC
BSF，BSR
SHLD，SHRD
条件转移的段内近转移格式
JECXZ
SETcc
IRETD

3. 486 提供的新指令

BSWAP
XADD
CMPXCHG

4. Pentium 提供的新指令

CMPXCHG8B

第 6 章 子程序结构

【教学目的】

通过本章学习，使学生掌握子程序设计的基本方法，并能够牢牢掌握过程定义伪操作和利用堆栈传送数据的方法。学习并了解多模块程序设计的方法。

【重点难点】

过程定义伪操作、利用堆栈传送数据、多模块程序设计，编写含子程序的汇编语言程序和上机操作。

【课 时 数】

6 学时。

1.子程序又称为过程，它相当于高级语言中的过程和函数。

2.模块化程序设计是按照各部分程序所实现的不同功能把程序划分成多个模块，各个模块在明确各自的功能和相互的连接约定后，就可以分别编制和调试程序，再把它们连接起来，形成一个大程序。

6.1 子程序的设计方法

6.1.1 过程定义伪操作：用在过程(子程序)的前后，使整个过程形成清晰的，具有特定功能的代码块。

伪操作格式：过程名(procedure name) PROC 属性(Attribute)

⋮

过程名(procedure name) ENDP

说 明：①.其中过程名(procedure name)为标识符，它又是子程序入口的符号地址。

②.属性(Attribute)是指类型属性，它可以是 NEAR 或 FAR。用户对过程属性的确定原则很简单：调用程序和过程在同一个代码段中则使用 NEAR 属性；不在同一个代码段中则使用 FAR 属性。主过程 MAIN 应定义为 FAR 属性。

6.1.2 子程序的调用和返回：80x86 的 CALL 和 RET 指令完成的就是调用和返回功能。

6.1.3 保存和恢复寄存器：就是保护现场和恢复现场。在一进入子程序后，就应该把子程序所需要使用的寄存器内容保存在堆栈中；在退出子程序前把寄存器内容恢复原状。

6.1.4 子程序的参数传送：调用程序和子程序之间的信息传送称为参数传送(或称变量传递或过程通信)。有以下几种参数传送方式：

1. 通过寄存器传送参数：这是最常用的使用最方便的一种方式，但参量很多时不能使用这种方法。

2. 如过程和调用程序在同一源文件(同一程序模块)中，则过程可直接访问模块中的变量

3. 通过地址表传送参数地址

4. 通过堆栈传送参数或参数地址：必须注意，子程序结束时的 RET 指令应使用带常数的返回指令，以便返回主程序后，堆栈恢复原始状态不变。

5. 多个模块之间的参数传送问题

(1). 外部符号

1). 局部符号和外部符号

a. 局部符号：在本模块中定义，又在本模块中引用的符号。

b. 外部符号：在某一模块中定义，而又在另一模块中引用的符号。

2). 与外部符号有关的两个伪操作：两个伪操作的使用必须相匹配。

a. PUBLIC 伪操作：在本模块中定义的符号(如变量、标号、过程名等)在提供给其它模块使用时必须要用 PUBLIC 定义该符号为外部符号。

伪操作格式：PUBLIC 符号(symbol)[，符号，…]

b. EXTRN 伪操作：在另一模块中定义，而要在本模块中使用的符号必须使用 EXTRN 伪操作。如符号为变量则类型 TYPE 应为 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE；如符号为标号或过程名则类型 TYPE 应为 NEAR 或 FAR。

伪操作格式：EXTRN 符号名(symbol name)：TYPE[，…]

(2). 多个模块之间的参数传送方法

1). 公共数据段方法：在每个模块中定义一个公共的数据段。用 COMMON 类型连接成覆盖的段。

2). 用外部变量方法：把变量定义为外部符号，可以供其它模块引用。如果程序中要访问的变量处于不同段时，就应动态地改变段寄存器的内容。

6.1.5 增强功能的过程定义伪操作

1. 从 MASM5.1 版开始为用户提供增强功能的过程定义伪操作

伪操作格式：procname PROC [attribute field][USES register list][, parameter field]

⋮

procname ENDP

说　　明：①.其中属性字段(attribute field)有以下几项组成：distance language type visibility prologue，每一项均为可选，各项之间用一空格或制表符分开。

Ⅰ.distance 就是 NEAR 或 FAR。

Ⅱ.language type 说明当该过程作为某高级语言程序的子过程时所用的高级语言。

Ⅲ.visibility 说明该过程的可见性，可用 Private 或 Public。如用 Private 则该过程的可见性只能是当前的源文件；如用 Public 则允许其他模块调用该过程(默认)。

Ⅳ.prologue 是一宏的名字,允许用户用宏来控制过程的入口和出口有关的代码。

②.USES 字段允许用户指定所需保存和恢复的寄存器表，MASM 将在过程入口自动生成 push 指令来保存这些寄存器，并在过程出口的 ret 指令前自动生成 pop 指令来恢复这些寄存器。

③.参数字段允许用户指定该过程所用参数，其格式为：identifier:type[,identifier:type] 其中 identifier 给出参数的符号名，type 给出参数的类型。参数之间用逗号隔开。

2. 增强功能的过程定义伪操作还可在过程中定义局部变量

(1). 局部变量是指在过程内部使用的变量。它是在过程被调用时在堆栈中建立的，在退出过程时被释放。它是以[BP-2]、[BP-4]、…的形式被访问的。

(2). MASM 规定，在过程内可以用 LOCAL 伪操作为局部变量申请空间

伪操作格式：LOCAL vardef [，vardef]

说　　明：①.其中变量定义可用的格式为：label label:type label [count]:type 其中第一个未指定类型者 MASM 将使用 WORD。type 可以指定任意合法的类型说明。第三种类型为用户申请数组提供了方便。

②.LOCAL 语句必须紧跟在过程定义伪操作之后，并在任何 80x86 指令或可以产生任何代码的 MASM 语句之前出现。它可以定义多个局部变量。

6.2 子程序的嵌套

1. 子程序的嵌套

(1). 子程序的嵌套：一个子程序作为调用程序去调用另一个子程序的情况称为子程序的嵌套。

(2). 嵌套深度：嵌套的层数称为嵌套深度。不受限制。

(3). 嵌套子程序的设计：没有什么特殊要求，除子程序的调用和返回应正确使用 CALL 和 RET 指令外，要注意寄存器的保护和恢复。如用堆栈传送数据要避免因堆栈使用中的溢出问题而造成子程序不能正确返回的错误。有两种可能发生堆栈溢出的情况：

1). 堆栈上溢：如堆栈已满，但还想再存入信息，这种情况称为堆栈上溢。

2). 堆栈下溢：如堆栈已空，但还想再取出信息，这种情况称为堆栈下溢。

2. 递归子程序：子程序自己调用自己的情况称为递归调用，这种子程序称为递归子程序。

6.3 子程序举例

例6.9 HEXIDEC 是一个把十六进制数转换成十进制数的程序。要求把从键盘输入的 0～FFFFH 的十六进制数转换为十进制数并从屏幕上显示出来。

例6.10 本例为一个简单的信息检索系统。在数据区里有 10 个不同的信息，编号为 0～9，每个信息包括 30 个字符。现要求编制一个程序：从键盘接收 0～9 之间的一个编号，然后在屏幕上显示出相应编号的信息内容。

例6.11 人名排序程序。先从终端键入最多 30 个人名，当所有人名都键入后，按字母上升的次序将人名排序，并在屏幕上显示已经排序后的人名。

例6.12 位串插入程序

第 7 章 高级汇编语言技术

【教学目的】

本章内容是汇编语言程序设计的拓展知识，可以提高设计汇编语言程序的技巧。属于介绍性内容，不作重点要求，不列入考试范围。有兴趣的同学可通过自学掌握更多的技巧和方法。

【重点难点】

宏定义、宏调用、宏展开。

【课 时 数】

2 学时。

7.1 宏 汇 编

7.1.1 宏定义、宏调用和宏展开

1. 宏：是源程序中一段有独立功能的程序代码。它只需要在源程序中定义一次，就可以多次调用，调用时只需要用一条宏指令语句就可以了。

2. 宏定义：是用一对伪操作 MACRO/ENDM 来实现的

伪操作格式：宏指令名(macro name) MACRO [哑元表(dummy parameter list)]
⋮ (宏定义体)
ENDM

说 明：①.其中 MACRO/ENDM 是一对伪操作。这对伪操作之间是宏定义体——是一组具有独立功能的程序代码。

②.宏指令名(macro name)给出该指令的名称，调用时就是用宏指令名来调用该宏定义。

③.哑元表(dummy parameter list)给出了宏定义中所用到的形式参数(或称虚参)，每个哑元之间用逗号隔开。

3. 宏调用：对宏指令的调用。必须先定义后调用。

宏调用格式：宏指令名(macro name) [实元表(actual parameter list)]

说 明：实元表(actual parameter list)中的每一项为对应哑元的实元，相互之间用逗号隔开。

4. 宏展开：当源程序被汇编时，汇编程序将对每个宏调用作宏展开。宏展开就是用宏定义体取代源程序中的宏指令名，而且用实元取代宏定义中的哑元。在取代时实元和哑元是一一对应的。一般说来，实元的个数应该和哑元相等，但汇编程序并不要求它们必须相等。另外汇编程序在展开的指令前加上“1”以示区别。

7.1.2 宏定义中的参数

1. 宏定义可以无变元。

2. 变元可以是操作码。

3. 变元可以是操作码的一部分，但在宏定义体中必须用&作为分隔符。

&——是一个操作符，它在宏定义体中可以作为哑元的前缀，展开时可以把&前后两个符号合并而形成一个符号，这个符号可以是操作码、操作数或是一个字符串。

4. 变元是 ASCII 串。

5. 宏定义的变元中使用的%操作符，汇编程序把跟在%之后的表达式的值转换成当前基数下的数，在展开期间，用这个数来取代哑元。

操作符格式：% 表达式(expression)

7.1.3 LOCAL 伪操作

伪操作格式：LOCAL 局部标号表(list of local labels)

说 明：①.局部标号(宏定义体内使用的标号)表(list of local labels)内的各标号之间用逗号隔开。汇编程序对 LOCAL 伪操作的局部标号表中的每一个局部标号建立唯一的符号(用??0000～??FFFF)以代替在展开中存在的每个局部标号。

②.LOCAL 伪操作只能用在宏定义体内，而且它必须是 MACRO 伪操作后的第一个语句，在 MACRO 和 LOCAL 伪操作之间还不允许有注释和分号标志。

7.1.4 宏定义中允许宏调用(宏嵌套)

1. 必须先定义后调用。
2. 宏定义体内不仅可以使用宏调用，也可以包含宏定义。

7.1.5 列表伪操作

1. .XALL 伪操作：不产生代码的语句在汇编后的 LST 清单中将不列出。(是汇编程序隐含的。)
2. .LALL 伪操作：在汇编后的 LST 清单中列出所有的语句(包括展开后的语句)。
3. .SALL 伪操作：在汇编后的 LST 清单中将不列出展开后的信息。
4. ;；——双分号伪操作：是 MASM 提供的伪操作，它用在宏定义或下一节要说明的重复语句中，前面使用双分号的注释，在宏展开时将不予展开。

7.1.6 宏库的建立与调用

1. 宏库：把自己编程中常用的宏定义建立成一个独立的文件，这个只包含若干宏定义的文件可称为宏库，通常用扩展名 MAC 或 INC 来表示。
2. 宏库的调用：当应用程序中需要用到宏库中的某些宏定义时，只需要在该程序的开始用 INCLUDE 语句说明为： INCLUDE [盘符：][\路径\]<宏库文件名>.MAC
汇编程序将把宏库中的所有宏定义都包含在应用程序中。

7.1.7 PURGE 伪操作：伪操作 PURGE 可以用来在适当的时候取消宏定义，以便恢复指令的原始意义。宏指令名可以与指令助记符或伪操作名相同，在这种情况下，宏指令的优先级最高，而同名的指令或伪操作就失效了。

伪操作格式：PURGE 宏指令名表 ；取消宏指令名表对应的宏定义

7.2 重 复 汇 编

有时汇编语言程序需要连续地重复完成相同的或者几乎完全相同的一组代码，这时可使用重复汇编。

7.2.1 重复伪操作

伪操作格式：REPT 表达式(expression)
┆ (重复块)
ENDM
说 明：①.其中表达式(expression)的值用来确定重复块的重复次数，表达式中如包含外部或未定义的项则汇编时指示出错。
②.重复伪操作并不一定要用在宏定义体内。

7.2.2 不定重复伪操作

1. IRP 伪操作
伪操作格式：IRP 哑元(dummy)，<自变量表(argument list)>
┆ (重复块)
ENDM
说 明：①.汇编程序把重复块的代码重复几次，每次重复把重复块中的哑元(dummy)用自变量表(argument list)中的一项取代，下一次用下一项取代，重复次数由自变量表中的自变量个数来确定。自变量表必须用尖括号括起，它可以是常数、符号、字符串等。
②.IRP 伪操作也并不一定要用在宏定义体内。

2. IRPC 伪操作
伪操作格式：IRPC 哑元(dummy)，字符串(string)(或<字符串(string)>)
┆ (重复块)

ENDM

说　　明：①.IRPC 和 IRP 类似，但自变量表必须是字符串(string)。重复次数由字符串中的字符个数确定。每次重复用字符串的下一个字符取代重复块中的哑元。可以不用尖括号括起。

②.IRPC 伪操作也并不一定要用在宏定义体内。

7.3 条 件 汇 编

1. 条件汇编：汇编程序根据条件把一段源程序包括在汇编语言程序内或者把它排除在外，这就是条件汇编。

2. 条件伪操作

伪操作格式：IFxx　自变量(argument)

┆　　}自变量满足给定条件汇编此程序块

[ELSE]

┆　　}自变量不满足给定条件汇编此程序块

ENDIF

说　　明：①.自变量(argument)必须在汇编程序第一遍扫视后就成为确定的值。

②.条件伪操作可以用在宏定义体内，也可以用在宏定义体外。也允许嵌套任意次。

③.条件伪操作中的“xx”表示条件如下：

IF expression	汇编程序求出表达式的值，如此值不为 0 则满足条件。
IFE expression	汇编程序求出表达式的值，如此值为 0 则满足条件。
IF1	在汇编程序的第一遍扫视期间满足条件。
IF2	在汇编程序的第二遍扫视期间满足条件。
IFDEF symbol	如符号(symbol)已在程序中定义，或者已用 EXTRN 伪操作说明该符号是在外部定义的，则满足条件。
IFNDEF symbol	如符号未定义或未通过 EXTRN 说明为外部符号，则满足条件。
IFB <argument>	自变量(argument)为空则满足条件。
IFNB <argument>	自变量(argument)不为空则满足条件。
IFIDN <arg-1>,< arg-2>	如果字符串<arg-1>和字符串< arg-2>相同，则满足条件。
IFDIF <arg-1>,< arg-2>	如果字符串<arg-1>和字符串< arg-2>不相同，则满足条件。

7.3.1 条件伪操作 IF 的使用举例

7.3.2 条件伪操作 IF1 的使用举例

7.3.3 条件伪操作 IFNDEF 的使用举例

宏定义中所使用的退出宏展开的伪操作：EXITM

伪操作格式：EXITM

7.3.4 条件伪操作 IFB 的使用举例

7.3.5 条件伪操作 IFIDN 的使用举例

第 8 章 输入/输出程序设计

【教学目的】

本章内容也是汇编语言程序设计的基本方法和具体应用，通过本章学习，使学生明确接口中数据的传送方式、输入输出程序设计的基本方法、无条件传送方式和查询式传送方式、中断操作的基本概念和实现中断功能的程序设计方法。

【重点难点】

数据传送方式、无条件传送方式和查询式传送方式、中断操作，中断程序设计。

【课 时 数】

4 学时。

计算机系统通过硬件接口以及 I/O 控制程序对外部设备进行控制，使其能协调地、有效地完成输入输出工作。能直接控制硬件的汇编语言就成了编写高性能 I/O 程序最有效的程序设计语言。

8.1 I/O 设备的数据传送方式

8.1.1 CPU 与外设

1. CPU 和外部设备的连接：通过硬件接口或控制器相连。

2. CPU 利用输入/输出指令(IN/OUT)与外部设备交换信息。这些信息有以下三种：

(1). 控制信息：CPU 输出到 I/O 接口，告诉接口和设备要做什么工作。

(2). 状态信息：CPU 从 I/O 接口输入，表示 I/O 设备当前的状态。

(3). 数据信息：双向传输，是 I/O 设备和 CPU 真正要交换的信息。

8.1.2 直接存储器存取 DMA 方式

1. 输入/输出的传送方式

(1). 程序直接控制 I/O 方式：CPU 通过执行程序不断读取并测试外设的状态，如果外设处于准备好状态(输入设备)或者空闲状态(输出设备)，则 CPU 执行输入指令或输出指令与外设交换信息。此为查询式传送方式。也可用无条件传送方式直接与外设交换信息。

(2). 中断传送方式：利用中断技术对输入/输出进行处理的方式。

(3). DMA 方式：直接存储器存取方式(或称为成组数据传送方式)。

(4). 通道控制方式：利用 I/O 协处理器的传送方式。

(5). I/O 处理机方式：利用专门的计算机进行 I/O 传送的 I/O 处理机方式。

2. DMA 方式

(1). DMA 控制器或接口一般包括四个寄存器：状态控制寄存器、数据寄存器、地址寄存器和字节计数器，这些寄存器能在信息传送之前进行初始化。每个字节传送后，地址寄存器增 1，字节计数器减 1。

(2). 计算机系统完成 DMA 传送的步骤

首先由接口发出 DREQ 信号，请求 DMA 控制器进行数据传送；

1). DMA 控制器向 CPU 发出 HOLD 信号，请求使用总线。

2). CPU 发出响应信号 HLDA 给 DMA 控制器，并将总线让出，DMA 控制器取得总线控制权。

DMA 控制器向接口发出 DACK（DMA 应答）信号，启动接口进行数据传送；

3). 传输数据的存储器地址通过地址总线发出。

4). 传输的数据字节通过数据总线传送。

5). 地址寄存器增 1。

6). 字节计数器减 1。

7). 如字节计数器未减 1 到 0，则转向第 3 步。

8). 否则，DMA 控制器撤销总线请求信号 HOLD，CPU 收回总线控制权，传送结束。

8.2 程序直接控制 I/O 方式

8.2.1 I/O 端口

1. I/O 端口：计算机为 I/O 接口中的寄存器分配的地址编码称为 I/O 端口。
2. I/O 端口的分类：一般接口都有控制端口、数据端口、状态端口。
3. 80x86 微机中，I/O 端口编址在一个独立的地址空间中，这个 I/O 空间允许设置 64K 个 8 位端口，32K 个 16 位端口。
4. 部分端口地址分配：书第 284 页的表 8.1 所示。

8.2.2 I/O 指令

1. IN——输入指令：输入数据和状态信息。

```
IN   AL，PORT     ；(AL)←(PORT)，端口直接寻址，(PORT) = 0～255
IN   AX，PORT     ；(AX)←(PORT+1):(PORT)，端口直接寻址，(PORT) = 0～255
IN   AL，DX       ；(AL)←((DX))，端口间接寻址，整个 I/O 地址空间(0～65535)
IN   AX，DX       ；(AX)←((DX)+1:(DX))，端口间接寻址，整个 I/O 地址空间(0～65535)
```

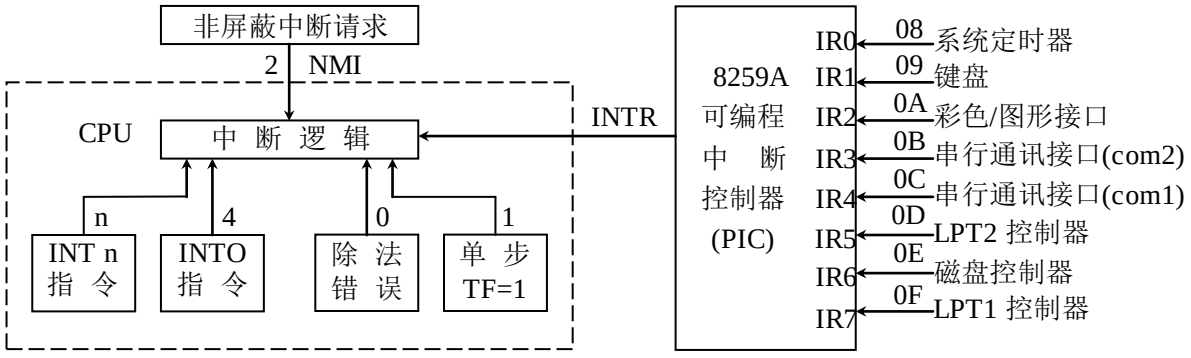
2. OUT——输出指令：输出数据和控制信息。

```
OUT  PORT，AL     ；(PORT)←(AL)，端口直接寻址，(PORT) = 0～255
OUT  PORT，AX     ；(PORT+1):(PORT)←(AX)，端口直接寻址，(PORT) = 0～255
OUT  DX，AL       ；(DX)←((AL))，端口间接寻址，整个 I/O 地址空间(0～65535)
OUT  DX，AX       ；((DX)+1:(DX))←(AX)，端口间接寻址，整个 I/O 地址空间(0～65535)
```

8.2.3 I/O 程序举例

8.3 中断传送方式

1. 中断：一种使 CPU 中止正在执行的程序而转去处理特殊事件的操作。
2. 中断源：引起中断的事件。
3. 外中断：由外部设备随机引起的一般由中断控制器控制的中断或某些处理器(8087/80287)引起的中断。
4. 内中断：由程序中安排的中断指令 INT 产生的中断，或由 CPU 的某些错误操作结果产生的中断。
5. 80x86 CPU 的中断源.：如下图所示。



8.3.1 8086 的中断分类

1. 软件中断(内中断)
(1). 中断指令 INT 引起的中断：CPU 执行完一条 INT n (n 不能为 0、1、2、4 的类型号)指令后，会立即产生中断，并且调用系统中相应的中断处理程序来完成中断功能。
(2). 处理 CPU 某些错误的中断(为处理运算结果的错误而设置)
1). 除法错中断：中断类型为 0。又称为除数为 0 中断。
2). 溢出中断：中断类型为 4。用 INTO 指令来中断发生溢出的算术操作，并把控制交给操作系统(此时要求 OF=1 并执行 INTO 指令)。如果未发生溢出，此时 OF=0，则 INTO 不会引起中断，继续执行下面的一条指令。
(3). 为调试程序(DEBUG)设置的中断

1). 单步中断：当标志位 TF 置为 1 时，每条指令执行后，CPU 自动产生类型 1 的中断——单步中断。

2). 断点中断：中断类型为 3。通常调试程序时，把程序按功能分成几段，然后每段设一个断点。当 CPU 执行到断点时便产生中断，这时程序员可以检查各寄存器及有关存储单元的内容。

2. 硬件中断(外中断)：外中断来自处理机的随机发生的外部条件。

(1). 外中断源：有非屏蔽中断(NMI 脚)和可屏蔽中断(INTR 脚)两大类。

(2). 外中断与 CPU 的连接

1). 非屏蔽中断接至 8086/8088 CPU 的 NMI 脚(17 脚)。

2). 可屏蔽中断通过 8259A 可编程中断控制器接至 8086/8088 CPU 的 INTR 脚(18 脚)。

(3). CPU 响应可屏蔽外中断(INTR)的条件

1). IF=1 才响应 INTR 请求的中断。

2). INTR 未被 8259A 的中断屏蔽寄存器(IMR)屏蔽。

a. IMR 的 I/O 端口地址为 21H。其 8 位对应 8 个外设的中断请求。

b. 某位为 0 允许该外设中断，某位为 1 则禁止该外设中断。

(4). CPU 响应非屏蔽外中断(NMI)是无条件的。该中断类型号为 2。

(5). 8259A 的中断结束命令 EOI 及中断命令寄存器(OCW2)

1). 8259A 的中断命令寄存器的端口地址为 20H。其内容为：

R	SL	EOI	0	0	L2	L1	L0

2). 中断结束命令 EOI 是使 8259A 的中断命令寄存器的第 5 位(EOI 位)置 1，清除当前中断服务寄存器 ISR 中对应的 ISn 位，以告诉 8259A 该中断已结束。

8.3.2 中断向量表

1. 中断类型号：每个中断都被安排一个以示区别的 8 位的类型编号称为中断类型号。80x86 中断系统能处理 256 种类型的中断，类型号为 0～0FFH。

2. 中断向量表：各个中断类型的处理子程序的入口地址表。存放于 00000～003FFH 的 1KB 单元中。低两位字节存放中断处理程序的偏移地址，高两位存放段地址。每个类型的中断向量地址占用 4 个字节的存储单元。

3. 内部中断的操作步骤

(1). 取中断类型号；

(2). 计算中断向量地址；

(3). 取中断向量：偏移地址→(IP)，段地址→(CS)；

(4). 转入执行中断处理程序；

(5). 中断返回到 INT 指令的下一条指令。

4. 设置中断向量：在检查或设置任何中断向量时，总是避免直接使用中断向量的绝对地址，而是使用 DOS 系统功能调用(21H)存取中断向量。

(1). 设置中断向量：把由 AL 指定的中断类型的中断向量 DS:DX 放置到中断向量表中。

1). 预置：AH=25H
AL=中断类型号
DS:DX=中断向量

2). 执行：INT 21H

(2). 取中断向量：把由 AL 指定的中断类型的中断向量从中断向量表中取到 ES:BX 中。

1). 预置：AH=35H
AL=中断类型号

2). 执行：INT 21H ；中断向量作为出口信息已在 ES:BX 中

8.3.3 中断过程：当中断发生时，由中断机构自动完成下列动作：

1. 取中断类型号 N

2. 标志寄存器(FLAGS)内容入栈

3. 当前代码段寄存器(CS)内容入栈

4. 当前指令计数器(IP)内容入栈

5. 禁止外部中断和单步中断(IF=0，TF=0)

6. 从中断向量表中取(4×N)中的字节内容送 IP，取(4×N+2)中的字节内容送 CS

7. 转中断处理程序。中断处理程序需注意：

(1). 如在执行中断处理程序中还允许中断，可用 STI 置 IF=1 来开中断；

(2). 注意保护现场和恢复现场；
(3). 中断处理程序返回时使用 IRET 指令。

8.3.4 中断优先级和中断嵌套

1. 8086 规定中断的优先级次序为

优先权高 内中断(除法错，INTO，INT)
↓ 非屏蔽中断(NMI)
可屏蔽中断(INTR)
低 单步中断

2. IBM PC 的可屏蔽中断(INTR)的优先级由高到低为：IR0，IR1，IR2，IR3，IR4，IR5，IR6，IR7。但是可以由 8259A 的中断命令寄存器(OCW2)的最高两位 D7、D6 位设置为循环的优先级。
3. 中断嵌套：正在运行的中断处理程序又被中断称为中断嵌套。
(1). 一般是高级中断嵌套低级中断，此时要求低级中断的处理程序中要有开中断指令 STI。
(2). 被同级或低级中断源嵌套，除了使 IF=1 外，还要求向 8259A 发中断结束命令 EOI。

8.3.5 中断处理程序

1. 中断准备：在主程序中设置。
(1). 设置中断向量
(2). 设置设备的中断屏蔽位
(3). 设置 CPU 的中断允许位 IF (开中断)
2. 中断响应：由硬件自动完成。
(1). 外设接口送中断请求给 CPU
(2). 当前指令执行完后，CPU 送中断响应信号给外设接口
(3). CPU 接收中断类型号
(4). 当前的 FLAGS，CS 和 IP 保存入栈
(5). 清除 TF 和 IF
(6). 中断向量送 IP 和 CS
3. 中断处理程序的编写方法与标准子程序类似，编写步骤如下：
(1). 保护现场，即保存寄存器内容
(2). 如允许中断嵌套，则开中断(STI) 使 IF=1
(3). 处理中断内容
(4). 关中断
(5). 送中断结束命令(EOI)给 8259A 的中断命令寄存器(OCW2)以清除 8259A 的 ISR 中对应的 ISn 位
(6). 恢复现场，即恢复寄存器的内容
(7). 返回被中断的程序(IRET，中断返回)

第 9 章 BIOS 和 DOS 中断

【教学目的】

本章内容是汇编语言程序设计的基本方法，通过本章学习，掌握在输入输出程序设计中大量使用的中断功能调用基本方法和具体应用、DOS 基本 I / O 功能调用、常用 BIOS 功能调用。

【重点难点】

DOS 基本 I / O 功能调用、常用 BIOS 功能调用，单个字符或字符串的输入输出功能调用。

【课 时 数】

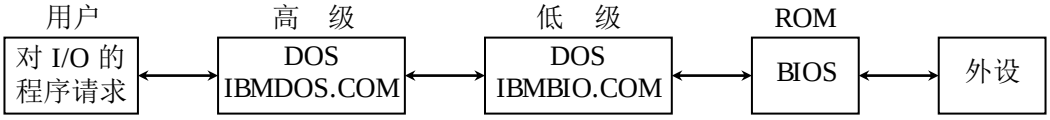
3 学时。

1. BIOS：Basic Input/Output System。在存储器系统中，从地址 FE000H 到 FFFFFH 的 8KB ROM 中装有 BIOS 例行程序。它提供了系统加电自检、引导装入、主要 I/O 设备的处理程序以及接口控制等功能模块来处理所有的系统中断。

2. DOS:Disk Operating System。由软盘或硬盘提供。它的三个模块中的两个模块与汇编程序设计密切相关。

(1). IBMBIO.COM：输入/输出设备处理程序。它提供了 DOS 到 ROM BIOS 的低级接口。

(2). IBMDOS.COM：一个文件管理程序和一些处理程序。它比 IBMBIO.COM 又高了一级。关系图如下：



3. BIOS 和 DOS 功能调用的不同点

(1). BIOS 能处理几乎所有的 I/O (只有发声等极少数不能处理)。它直接用输入/输出指令同端口打交道。

(2). DOS 能处理大多数 I/O。它是通过调用 BIOS 来同端口打交道的。级别高，使用方便，程序移植性好。但处理的数量少于 BIOS。

4. IBM PC 系统主要的中断类型

(1). BIOS 中断类型：见书第 316 页表 9.1。

1). CPU 中断类型：0～7

2). 8259A 中断类型：8～0FH

3). BIOS 中断类型：10～19H，40H

4). 用户应用程序：1BH，1CH，4AH

5). 数据表指针：1DH，1EH，1FH，41H，46H

(2). DOS 中断类型：见书第 316 页表 9.2。20～2FH，30～3FH 保留给 DOS。

5. DOS 功能和 BIOS 功能都通过软件中断调用。调用 DOS 或 BIOS 功能时，有以下几个基本步骤：

(1). 将调用参数装入指定的寄存器中；

(2). 如需功能号，把它装入 AH;

(3). 如需子功能号，把它装入 AL;

(4). 按中断号调用 DOS 或 BIOS 中断；

(5). 检查返回参数是否正确。

9.1 键盘 I/O

1. 键盘是计算机最基本的一种输入设备，用以输入信息，以达到人机对话的目的。

2. 键盘提供了三种基本类型的键

(1). 字符键：传送一个 ASCII 码字符给计算机。如 A～Z，0～9，%，$ 等。

(2). 扩展功能键：产生一个动作。如 Home，End，Enter 等。

(3). 组合功能键：改变其它键所产生的字符码。如 Alt，Ctrl，Shift 等。

9.1.1 字符码与扫描码

1. BIOS 的键盘中断：当 8259A 的 21H 端口第 1 位(D_1)为 0，就允许键盘中断，在键盘上“按下”或“放开”一个键时都会产生一个类型号为 9 的中断。并转入 BIOS 的键盘中断处理程序。

2. 扫描码：键盘中断处理程序从 8255 的 60H 端口读取一个字节，该字节的低 7 位就是键的扫描码。键盘上的每一个键对应一个扫描码，从 01H～51H。

(1). 通码：“按下”时取得的字节为通码，D_7 位=0。

(2). 断码：“放开”时取得的字节为断码，D_7 位=1。

3. 字符码：BIOS 键盘处理程序将所取得的扫描码转换成相应的字符码。大部分键的字符码为 ASCII 码，没有 ASCII 码的键其字符码为 0 或一个指定的操作(如屏幕打印等)。

4. BIOS 数据区的键盘缓冲区 KB_BUFFER ：一个先进先出(FIFO)并具有双指针的循环队列。存放转换成的字符码和扫描码(扫描码存放在字的高 8 位上)。当 CPU 想要得到键盘输入时，就调用 BIOS 键盘例行程序，它按其接收时的次序从缓冲区取出字符和扫描码，回送给 CPU。键盘缓冲区结构如下：

```
0040:001A   BUFF_HEAD       DW   ?           ; 键盘缓冲区的首地址指针
0040:001C   BUFF_TAIL       DW   ?           ; 键盘缓冲区的末地址指针
0040:001E   KB_BUFFER       DW   16 DUP (?)  ; 16 个输入量的空间
0040:003E   KB_BUFFER_END   LABEL  WORD
```

9.1.2 BIOS 键盘中断：INT 16H 调用(从键盘缓冲区输入)。

1. AH=0：从键盘读一字符。此时一直要等到键盘缓冲区有输入才返回。返回时 AL=字符码，AH=扫描码。

2. AH=1：读键盘缓冲区的字符。如 ZF=1 则键盘缓冲区空；ZF=0 则 AL=字符码，AH=扫描码。

3. AH=2：取键盘状态字节。AL=键盘状态字节(KB_FLAG：0040:0017H 单元)。

(1). 键盘状态字节 KB_FLAG：内存的 0040:0017H 单元，由类型 9 的硬件键盘中断置入键盘的对应状态到该单元。低 4 位是 Alt、Ctrl、Shift 左、Shift 右的标志位，这 4 位在相应键按下时置位，该键一抬起即复位。KB–FLAG 的高 4 位是 Ins、Caps Lock、Num Lock、Scroll Lock 键的标志位，这些位在相应键奇次按下时置位，偶次按下时复位。具体含义如右：

KB_FLAG：0040:0017H 单元

D7	D6	D5	D4	D3	D2	D1	D0

- D0：1=按下右移键 Right Shift
- D1：1=按下左移键 Left Shift
- D2：1=按下控制键 Ctrl
- D3：1=按下交替键 Alt
- D4：1=Scroll Lock 状态已变换
- D5：1=Num Lock 状态已变换
- D6：1=Caps Lock 状态已变换
- D7：1=Insert 状态已变换

(2). 键盘的其它标志区：

Ins'	Caps'	Num'	Scroo'	Hold			

1). KB–FLAG1：0040:0018H 单元如右。其中的高 4 位是 Ins、Caps Lock、Num Lock、Scroll Lock 锁定功能键的动态(临时)标志位，即这些键按下时该位置 1，键释放即被置 0。这 4 位动态标志位用于对这些键本身进行处理，而 KB–FLAG 中的这 4 位静态标志位反映了它们的锁定状态，用于控制其它键的键意确认。KB–FLAG1 中的 D3 位为 Hold–State 保持状态位，Ctrl+Num 键按下时该位置 1；非 Num 键的任一键按下就使其复位。暂停过程就是将该位置 1 后进入一个无限循环过程，判断该位是否变为“0”，一但为“0”，即退出循环。

2). ALT–INPUT：0040:0019H 单元。此单元不是标志单元，而是寄存单元。在按着 Alt 键不释放的条件下，连续按下小键盘(右区)的数字键×××(1~255)时，此 1~3 位的十进制数的二进制编码即在此生成。当 Alt 键释放时，第一字节为 ALT–INPUT 单元中的代码、第二字节为 00H 的两个字节存入键盘缓冲区，ALT–INPUT 单元又被清 0。若打入的数字键大于 255 数值，则以 256 为模产生结果。这是一种直接在键盘缓冲区建立所期望的 ASCII 代码的一种方法，第二字节为 00H 就与其它键所建立的 ASCII 代码相区别。

3). BIOS–BREAK：0040:0071H 单元。此单元是中止或称为间断标志单元。Ctrl+Scroll 键按下时，该标志字节置为 80H，否则为 00H，Ctrl+Scroll 键的按下还清除键盘缓冲区，并以两个全 0 字节为其 ASCII 码，存入键盘缓冲区。此标志和特征码均可被其他软件用来证实“间断”键的按下。但是 ROM–BIOS 并不直接提供间断功能，在键盘中断服务程序中，对该组合键的按下提供了一条 INT 1BH 指令。在 ROM–BIOS 中类型 1BH 中断的服务程序只是一条中断返回指令 IRET，其它什么都不做。其它软件只需将其“间断”功能程序的入口地址重新写入中断向量表的 1BH×4 的位置中去即可。因而即对其它软件提供了查询处理的方便，也提供了中断处理的方便，用以实施“间断”功能。

4). RESET–FLAG：0040:0072H 单元。是两字节的复位标志。当 Ctrl+Alt+Del 组合键按下时，就以 1234H 代码填入，然后转移到复位初始化程序，这就是热启动复位。

5). Shift+Prtsc 组合键：是打印屏幕的特殊操作。按下时由键盘中断服务程序执行软件中断指令 INT 5 产生的。这个组合键既不向键盘缓冲区存代码，也不建立标志，但是 INT 5 软件中断服务程序使用 STATUS–BYTE 标志单元(0050:0000)，使在打印屏幕过程中再按下 Shift+Prtsc 键将不被理睬。

9.1.3 DOS 键盘功能调用：(用 BIOS 键盘缓冲区输入)设置 AH，执行 INT 21H 的 DOS 功能调用实现从键盘输入字符。

1. 单字符输入：输入的字符在 AL 寄存器中返回。

(1). AH=1：从键盘输入一个字符并回显在屏幕上，并检验是否按下了 Ctrl_Break 键，如按下则自动调用 INT 23H 并结束程序。返回时 AL=字符。

(2). AH=6：读键盘字符，入口时 DL=0FFH。如果准备好，AL=字符，未准备好，AL=0。

(3). AH=7：从键盘输入一个字符不回显，也不检验是否按下了 Ctrl_Break 键。返回时 AL=字符。

(4). AH=8：除不回显以外，同功能 1。

(5). 注意：若要求程序能接收功能键及具有键扩展码的组合键必须进行两次 DOS 功能调用，第一次回送 00，第二次回送扫描码。

2. 输入字符串：AH=0AH。从键盘输入一串字符并存入用户定义的缓冲区中(DS:DX=缓冲区首地址)。

```
BUF   DB   50
      DB   ?
      DB   50 DUP(?)
```

(1). 缓冲区的第一字节给出能输入的最大字符数。由用户程序给出。如右：

(2). 第二字节为调用后实际输入的字符数。由功能 0AH 调用填入，不含回车符。

(3). 第三字节开始为调用后实际输入的字符串。由功能 0AH 填入，不含回车符。

(4). DOS 调用后，保持 DS:DX=缓冲区首地址不变。

3. 清除键盘缓冲区：AH=0CH。清除后并再调用一种键盘功能。被调用的键盘功能号(只能是 1、6、7、8、0AH)作为入口信息放在 AL 寄存器中。

4. 检验键盘状态：AH=0BH。返回时 AL=0FFH 表示有键入，AL=00 则无键入。返回后执行下一条指令。

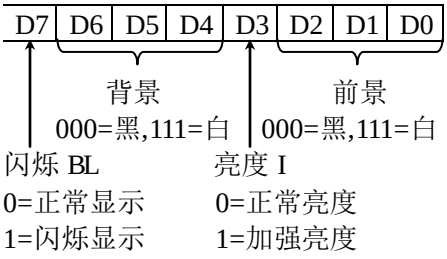
9.2 显示器 I/O

1. 显示器适配器：用于计算机和显示器连接的接口电路板，也称为显示卡。在 IBM PC 中有单色显示及并行打印机适配器，彩色图形监视器适配器两种。

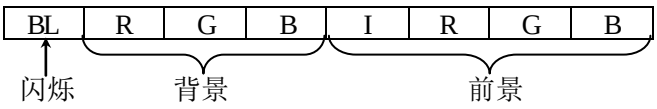
2. 象素：80 列 25 行共 2000 个网格的位置为象素。每个象素可显示一个字符，且在存储器中都有一个字映像。屏幕行号为 0～24(0～18H)，列号为 0～79(0～4FH)。

3. 属性：对应显示屏上的每个字符在存储器中由连续的两个字节表示，一个低字节表示字符的 ASCII 码，另一个高字节保存字符的属性。

4. 文本方式：在屏幕上处理字母、数字以及一些字符图形称为文本方式。



9.2.1 字符属性



1. 单色字符显示：单色显示属性字节如右上图：

2. 彩色字符显示：彩色显示属性字节如右下图。

3. 显示存储器：屏幕上某一字符位置在显示存储器(又称为显存)中的偏移地址可由下列公式算出：

Char_offset＝Page_offset＋((row×width)＋column)×byte

其中 Page_offset 是页偏移地址，width 是每行可显示的字符数，在 25×80 的字符显示方式下，width=80，byte 是表示一个字符所用的字节数，在字符显示方式下 byte=2，row 和 column 是相对于屏幕左上角位置(0，0)的行列坐标。

9.2.2 BIOS 显示中断：INT 10H 的 0～0FH 及 13H 共 17 个功能。部分见书第 328 页的表 9.8 所示。

1. 控制光标：光标不是 ASCII 字符，它由硬件产生和控制。

(1). AH=1：控制光标行的开始(CH 的低 4 位)和结束(CL 的低 4 位) (即大小)，显示光标(CH4=0)和关闭(CH4=1)。

(2). AH=2：设置光标位置。DH=行号，DL=列号，BH=页号。对单显 BH=0。

2. 读光标位置：AH=3，读光标位置。BH=页号。出口信息为 DH=行号，DL=列号；CH 和 CL 中的低 4 位为光标大小。

3. 选择显示页：AH=5，AL=页号。可由程序确定显存中的显示区域。

4. 清屏和卷屏

(1). AH=6：使屏幕内容或窗口内容上卷指定的行。该功能需设置 7 个入口参数，AL=0 时清屏或清窗口。

(2). AH=7：使屏幕内容或窗口内容下卷指定的行。该功能需设置 7 个入口参数，AL=0 时清屏或清窗口。

5. 字符显示

(1). AH=9：在光标位置显示字符及属性。BH=页，AL=字符，BL=属性，CX=字符重复次数。光标回到原位置。

(2). AH=0AH：在光标位置只显示字符。BH=页，AL=字符，CX=字符重复次数。光标回到原位置。

(3). AH=8：读光标位置的字符及属性。BH=页。返回时 AL=字符，AH=属性。

6. 彩色和字符串显示

(1). 在编写字符显示程序时，彩色显示和单色显示类似。注意设置属性值。

(2). AH=13H 功能显示字符串有 4 种方式，前两种方式(AL=0，1)要指定整个显示字符串的属性，后两种方式(AL=2，3)要指定每个字符的属性。

9.2.3 DOS 显示功能调用(INT 21H 的部分功能)

1. 显示单字符

(1). AH=2：显示一个字符并检验 Ctrl_Break。DL=(显示)字符，光标跟随字符移动。

(2). AH=6：直接控制台 I/O，不检验 Ctrl_Break。

1). DL=0FFH：为输入字符，AL=输入字符；

2). DL=字符：为输出并显示一个字符，光标跟随字符移动。

2. AH=9：显示字符串。DS:DX=字符串的首地址，字符串必须以“$”为结束符，光标跟随字符串移动。在“$”前可加回车(0DH)换行(0AH)符。

9.3 打印机 I/O

1. 打印机：是计算机的主要硬拷贝设备。按照印字原理分为字模式、针式、喷墨式、热转印式、激光式、LED 式、LCS 式、荧光式、电灼式、磁式和离子式等多种。

2. 打印机接口：有串行口和并行口两种。IBM PC 系列机使用并行口，且打印机为 Centronics 型的 36 针插座，计算机上的并行口为 D25 型的插座。两者需要用专用的打印电缆连接。

3. 打印字符/图形要求软件将字符/图形的输出转化为打印机的控制码，这些软件通常称为打印机驱动程序。

4. 不同的打印机具有不同的控制字符，需要查阅该打印机的说明书。

9.3.1 DOS 打印功能：INT 21H 的功能 5 (AH=5)，打印一个放在(DL)寄存器中的字符。(可送一些控制字符码：如回车、换行、换页等。)

9.3.2 打印机的控制字符

1. 标准控制字符：有空格——08H，水平 TAB(横表)——09H，换行——0AH，垂直 TAB(纵表)——0BH，换页——0CH，回车——0DH。(TAB 功能在某些打印机中没有。)

2. 特殊的打印命令

(1). 部分特殊打印命令：设置紧缩方式——0FH，设置扩展方式——0EH，取消紧缩方式——12H，取消扩展方式——14H。

(2). 与 ESC(1BH)字符一起使用的命令：有很多，需要阅读打印机手册。

(3). 向打印机发送命令码的格式

1). 在数据区中定义命令码；

2). 直接用命令方式发送。

打印机的状态字节

D7	D6	D5	D4	D3	D2	D1	D0
打印机忙	应答位	纸出界	选择打印机	I/O 错	未用	未用	超时

9.3.3 BIOS 打印功能(INT 17H)

1. AH=0：打印一个由 AL 指定的字符。并回送状态信息到 AH 中。

调用时需用 DX 指定打印机号(0、1、2 号)。

2. AH=1：初始化由 DX 指定号(0、1、2 号)的打印机。并回送状态信息到 AH 中。

3. AH=2：读由 DX 指定号(0、1、2 号)的打印机的状态信息到 AH 中。

4. 打印机的状态字节如上页图所示。

9.4 串行通信口 I/O

1. 计算机传输数据有并行和串行两种方式。

2. 串行通信采用两种方式：同步方式和异步方式(书中将同步和异步方式弄反了)。

9.4.1 串行通信接口：计算机利用异步通讯适配器和调制解调器(MODEM)通过 RS-232 串行接口与外进行远距离通信。近距离通信不用 MODEM。

1. 串行通信基础

(1). 在异步通信方式中，每个字符都需要加上起始位、校验位和终止位，此过程称为“组帧(framing)”。

(2). 串行通信的数据传输率用 bps(bits per second)来表示，在计算机中就是波特率。

2. RS 232 串行通信接口：为了兼容各厂家生产的数据通信设备，1960 年电子工业协会(EIA)制定了 RS-232 接口标准。

3. IBM PC 通信端口

(1). IBM PC 机提供两个串行接口 COM1 和 COM2。80x86 兼容机可以连接 4 个通信端口，它们的编号为 COM1~4(相应的 BIOS 编号为 COM0~3)。

(2). 如果微机系统设置了 COM 端口，则设置的每个 COM 端口的 I/O 地址就写到 BIOS 数据区的 0040：0000~0040：0007 字节，每个 COM 地址占用 2 个字节。

9.4.2 串行口功能调用

1. DOS 串行通信口功能

(1). 使用 DOS 命令 MODE 可以设置串行通信参数，如数据的字长、波特率、校验位和终止位数。设置串行通信参数命令的一般格式为：

MODE COMm：b，p，d，s

这里 m 表示 COM 的端口号(1~4)；b 是波特率，用波特率数的高两位数字来表示；p 是校验位(N 为无校验，O 为奇校验，E 为偶校验)；d 表示数据的字长(5，6，7，8 位、默认值为 7 位)；s 是终止位的位数(1，1.5 或 2 位)。

(2). DOS 手册中称串行通信口为辅助设备。DOS 的 INT 21H 调用，对 COM1 操作。

1). AH=3：从 COM1 读一个字符到 AL 中。

2). AH=4：将 DL 中的字符传送给 COM1。

2. BIOS 串行通信口功能：INT 14H 调用。

(1). AH=0：初始化串行口，AL=初始化参数(图 1)，DX=通信口号(COM1 为 0 号，COM2 为 1 号)。在 AH 中返回通信口状态(图 2)，AL 中返回调制解调器状态(图 3)。

(2). AH=1：向串行口(DX 号)写 AL 指定的字符。写成功 AH_7=0，AL 不变；写失败 AH_7=1，$AH_{6\sim0}$=通信口状态。

(3). AH=2：从串行通信口(DX 号)读字符。读成功 AH_7=0，AL 为所读字符；读失败 AH_7=1，$AH_{6\sim0}$=通信口状态。

(4). AH=3：取串行通信口(DX 号)状态。在 AH 中返回通信口状态，AL 中返回调制解调器状态。

图 1 串行通信口初始化参数

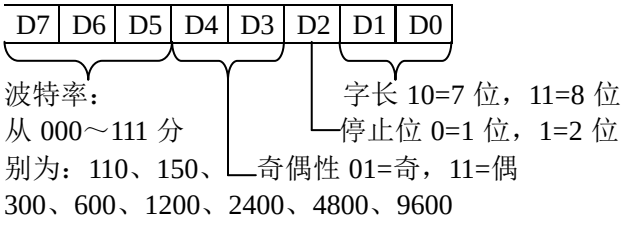
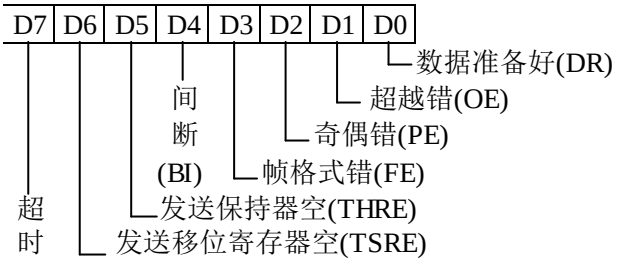


图 2 串行通信口状态字节



第 10 章 图形与发声系统的程序设计

【课 时 数】

0 学时。

10.1 显示方式

10.1.1 显示分辨率

1. 显示分辨率：包括字符分辨率和像素分辨率，分别表示显示器在水平和垂直方向上所能显示的字符数和像素(pixel)数。
2. 目前较流行的彩色图形适配器：增强型图形适配器(enhanced graphics adapter，EGA)和视频图形阵列适配器(video graphics array，VGA)。
3. 较新的显示卡有：XGA、Super VGA (SVGA)等。

10.1.2 BIOS 设置显示方式

1. 显示方式分为两类：文本方式和图形方式。文本方式主要用于字符文本处理，图形方式又称为所有点可寻址(all-point-addressable)方式。
2. 显示方式的设置：BIOS 中断 INT 10H 的功能 0 (AH=00H)。利用 AL 中的值设置不同的显示方式。
3. 设备检验：BIOS 中断 INT 11H。在 AX 中返回设备的配置情况，即在 AX 中返回设备标志(Equipment Flag)。其中的 4、5 两位为显示器的配置。设备标志的存储单元地址为：0040:0010。
4. 另一种确定适配器的方法是，先检验 VGA，再检验 EGA，最后确定是 CGA 还是 MDA。
5. 单纯为了获取或改变当前显示方式，还有一种方法是调用 INT 10H 显示功能 0FH，该功能将当前显示方式返回在 AL 中。

10.2 视频显示存储器

在显示器上要显示的信息(文本或图形数据)都存放在称作视频显示 RAM(VDR)的存储器中，VDR 也称为显示缓冲区。程序可以通过指令对 VDR 读取和写入。

10.2.1 图形存储器映像

在 EGA 和 VGA 的图形方式下，像素的存取是采取一种位映像(bit-mapped)的方式。对视频存储器的一个地址进行读写操作，将会从 4 个并行的位面(bit planes)存储 4 个字节的数据，这 4 个位面的存取由 CPU 根据对相应的锁存寄存器的设置来决定。

1. EGA 视频存储器：支持 640×350 像素、最多可显示 64 种颜色，但同时在屏幕上可显示的颜色数只有 16 种。
2. VGA 视频存储器：支持 640×480 像素、最多可同时显示 256 种颜色。
3. SVGA 和其他显示适配器：支持 800×600、1024×768、1024×1024 等分辨率，有 16 位、24 位和 32 位真彩色等。

10.2.2 数据到颜色的转换

EGA/VGA 中使用了一种颜色编码技术，它包括在 EGA 上使用 2 位的颜色代码，在 VGA 上使用 6 位的颜色代码。

10.2.3 直接视频显示

用于 EGA 和 VGA 标准的位面结构的一个重要特征是，在指定的图形方式中，通过设置位面的存储位来确定每个像素的显示状态。在所有位面结构的显示方式下，地址映像操作都需要计算两个值：一个是含有该像素存储位的字节地址，另一个是分离像素位所需要的掩码(mask)。

1. 字节级映像操作：有的图形应用程序需要对屏幕上的一块区域进行控制，比如在某一区域显示彩色阴影、开窗口、画条块或其他对称的图形，这时只需按字节对像素进行读写操作，而无需细分到位。
2. 位级映像操作：当 EGA 或 VGA 图形程序读写一个屏幕像素时，必须首先获得与这个像素相对应的字节地址，然后计算位掩码，并用掩码把该像素与映射在同一字节地址中的其他 7 个像素分离出来。确定位掩码一般有两种方法：
(1). 一种方法是通过对一个基本位模式 10000000 右移来获得掩码，移位次数是 X 坐标除以 8 得到的余数。
(2). 另一种方法是用这个余数作为索引值，去查找一个包含有 8 个掩码的表。索引 0 对应掩码 10000000B，索引 1 对应掩码 01000000B，以此类推。

10.3 EGA/VGA 图形程序设计

在图形方式下，可以利用 BIOS INT 10H 功能或采用直接存储器映像的方法对屏幕上像素进行读写和处理。在目前 BIOS 提供图形处理功能还相对弱的情况下，许多图形程序都采用了直接视频显示的方法。

10.3.1 读写像素

当 EGA/VGA 为图形方式时，BIOS 仅有两个例程用于读写，即 INT 10H 的 AH=0CH(写像素)，AH=0DH(读像素)，这两个功能可以很方便地读或写一个像素点到视频存储器。调用时程序员提供颜色、页号、行号和列号。

1. 读模式：EGA 和 VGA 提供两种读模式
(1). 读模式 0：是一种默认模式，读模式 0 可以单独读取某个位面的字节，这在位面和主存或磁盘间传输数据时非常有用。
(2). 读模式 1：CPU 将 8 个像素的值读入锁存器。这 8 个像素值会与颜色比较寄存器作比较，相符合者置为 1，反之置为 0，然后把比较的结果送到 CPU。
2. 写模式：EGA 有 3 种写模式，VGA 有 4 种写模式，选择写模式通过设置图形控制器的模式选择寄存器来实现。
(1). 写模式 0：是 EGA 和 VGA 的默认模式，在写模式 0 中，所写入的 CPU 数据可以更新任何一个或是全部的位面，同时，还可以与一个事先定义好的值进行逻辑运算，以更新锁存器中的 8 个像素或其中任一个像素。
(2). 写模式 1：能把先前读入锁存寄存器的数据直接复制到位面中。
(3). 写模式 2：是写模式 0 的简化方式，但它比写模式 0 的执行速度快。
(4). 写模式 3：是 VGA 系统独有的，像素值是由锁存器中的像素值和置位/重置寄存器的值合并后产生的。

10.3.2 图形方式下的文本显示

INT 10H 的功能 9 是 BIOS 提供的唯一能用于图形方式的字符显示功能。另外 INT 10H 的功能 2 也能在图形方式下设置字符显示位置。

10.3.3 彩色绘图程序

许多绘图程序的技术，都是从处理视频缓冲区(显存)中的像素做起的。一旦计算出一个指定像素在显存中的地址，那么读写一个像素值就是很简单的了。由于很多绘图程序需要处理的像素数量较大，因此高效快速是绘图程序很重要的一个指标，这也是绘图程序大多采用直接视频显示，而不用 BIOS 视频例程的原因。即使这样，编写简单而又高效的图形程序仍有许多设计技巧值得探讨。

10.3.4 动画显示技术

1. 计算机动画是利用计算机图形显示技术来模仿物体活动的效果，一般分为两种类型：逐帧(frame-by-frame)动画和实时(real-time)动画。
(1). 逐帧动画技术的最重要的用途是设计图形系列以建立用不同介质表现的动画图像。
(2). 实时动画是直接在终端上显示动画程序执行的结果。
2. 屏幕物体的动画效果，经常通过几何变换来产生，最简单的几何变换有平移、旋转和比例变换，复杂的动画通过组合两个或更多的变换来完成。在所有情况下，变换都是以一个新的图像代替先前的

图像来实现。

3. 擦除和重画屏幕物体的周期有几种实现的方法：

(1). 最直接的一种方法是在显示图形之前，把图形将要占据的屏幕部分的背景图像保存下来，当要擦除图形时，再把保存的背景图像重新显示出来。

(2). 另一种擦除屏幕图像的方法是基于 XOR 操作的方法，在动画程序中，XOR 操作提供了一种连续显示图形和擦除图形的方法。

(3). 计算机动画是由一定速率的显示操作和擦除操作完成的，有时这个技术也被称作定时脉冲动画(timed-pulse animation)。在定时脉冲动画中，理想的重画速率应不小于每秒 24 幅图像。

10.4 通用发声程序

10.4.1 可编程时间间隔定时器 8253/54

1. 在 8253/54 定时器内部有 3 个独立工作的计数器：Counter0、Counter1 和 Counter2，每个计数器都分配有一个端口地址，分别为 40H、41H 和 42H。8253/54 内部还有一个公用的控制寄存器，端口地址为 43H。

2. 对 8253/54 编程时，先要设定控制字，以选择计数器，确定工作模式和计数值的格式。每个计数器由三个引腿(CLK、GATE 和 OUT)与外部联系。

3. 8253 控制寄存器的格式和含义

(1). 8253 控制寄存器的格式

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RW_1	RW_0	M_2	M_1	M_0	BCD

(2). 8253 控制寄存器的含义

1). D_0(BCD)：用来设置计数值格式。

a. BCD=1：计数值为 BCD 码格式；

b. BCD=0：计数值为二进制格式。

2). $D_3D_2D_1$($M_2M_1M_0$)：模式选择。每个计数器都可用 6 种模式之一工作。

a. $M_2M_1M_0$=000：模式 0；

b. $M_2M_1M_0$=001：模式 1；

c. $M_2M_1M_0$=x10：模式 2；

d. $M_2M_1M_0$=x11：模式 3；

e. $M_2M_1M_0$=100：模式 4；

f. $M_2M_1M_0$=101：模式 5。

3). D_5D_4(RW_1RW_0)：读/写指示位。

a. RW_1RW_0=00：对计数器进行锁存操作，使当前计数值在输出锁存器中锁存，以便读出；此时 D_3~D_0 位不被改变。

b. RW_1RW_0=01：只读/写低 8 位字节；

c. RW_1RW_0=10：只读/写高 8 位字节；

d. RW_1RW_0=11：先读/写低 8 位字节；再读/写高 8 位字节。

4). D_7D_6(SC_1SC_0)：选择要写入的计数器。

a. SC_1SC_0=00：选择写入计数器 0；

b. SC_1SC_0=01：选择写入计数器 1；

c. SC_1SC_0=10：选择写入计数器 2；

d. SC_1SC_0=11：无意义。

4. IBM PC 机中 8253/54 的编程使用

(1). 计数器 0 作为定时器为系统日时钟提供计时基准。

(2). 计数器 1 作为定时器使用，其输出脉冲作为 DRAM 刷新的定时信号。

(3). 计数器 2 用来控制扬声器发声。在 IBM BIOS 中有个 BEEP 子程序，它在模式 3 下，能产生频率为 896Hz 的声音。

10.4.2 扬声器驱动方式

1. 可编程外围接口芯片(PPI——8255 或 8255A)： PC 机上的大多数输入/输出都是由系统插件板上的

PPI 管理的。8255 有 3 个 8 位寄存器，A 口、C 口用于输入，端口号为 60H、62H；B 口用于输出，端口号为 61H。其中 PB_0 控制定时器 2，PB_1 控制扬声器与定时器 2 的接通与否。

2. 两种发声方法

(1). PB_0=0，PB_1=1/0 (切换)可以发声。

(2). PB_0=1，PB_1=1 用定时器 2 的振荡频率控制发声。ROM BIOS 的 BEEP 子程序：根据 BL 中给出的值控制定时器 2 的输出频率来发声。

10.4.3 通用的发声程序：频率任意，持续时间任意。

1. 该程序由用户在 DI 寄存器中输入 19～65535Hz 的频率。由 n=F/f(F=1192576Hz)计算定时器 2 的计数值 n。

2. 发声的持续时间(该频率声音的时间) / 10ms 送 BX 寄存器。

3. 通用发声程序 GENSOUND (例 10.13)。

10.4.4 80x86 PC 的时间延迟

在 IBM PC AT BIOS 中的 WAITF 子程序，就是一个产生 N×15.08μs 时间延迟的程序。调用 WAITF 子程序时，CX 寄存器必须装入 15.08μs 的倍数 N。利用 WAITF 子程序能获得任意的延迟时间，而再不必考虑 CPU 的信号和工作频率。(修改后的通用子程序改名为 SOUNDF。)

10.5 乐曲程序

10.5.1 音调与频率和时间的关系

1. 乐曲是按照一定的高低、长短和强弱关系组成的音调。

2. 组成乐曲的每个音符的频率值和持续时间是乐曲程序发声所需要的两个数据。

3. 音符的持续时间是根据乐曲的速度及每个音符的节拍数来确定的。

10.5.2 演奏乐曲的程序

利用上节的发声子程序编写乐曲演奏程序。编写步骤如下：

1. 为演奏的乐曲定义一个频率表和一个节拍表。

2. 分别将两个表的偏移地址放入 SI 和 BP。

3. 从表中取出音符的频率放入 DI，取出音符的持续时间(实际上是 10ms 的倍数)放入 BX。

4. 调用 SOUNDF 子程序发出音调。

10.5.3 键盘控制发声程序

利用键盘来模拟钢琴弹奏乐曲。发声程序的又一应用举例。

第 11 章 磁盘文件存取技术

【教学目的】

本章内容也是汇编语言程序设计的基本方法，通过本章学习，掌握用中断系统功能调用进行文件代号法存取文件的基本方法和具体应用，学习有关磁盘记录方式的相关知识和内容，掌握路径和 ASCIZ 串、文件代号等知识点。

【重点难点】

磁盘目录及文件分配表、路径名和 ASCIZ 串、文件代号、磁盘文件读写、字符设备的文件代号式 I/O。

【课 时 数】

3 学时。

1. 外部设备的分类(两类)
  (1). 字符设备：键盘、显示器、打印机、串行口等。
  (2). 大容量存储设备：磁带、软磁盘、硬磁盘等。
2. 文件：存放在磁盘上的程序或数据。能用各种程序语言处理的一般是 ASCII 码文件，而二进制文件需用其它方式处理。
3. 文件在磁盘中的存取方式
  (1). 顺序存取方式：把文件划分成记录，然后从文件的首记录到末记录顺序进行存取。
  (2). 随机存取方式：随机地存取文件中的任何记录。
  (3). 随机分块存取方式：随机地对记录块(若干个记录)进行存取。
  (4). 文件代号式磁盘存取方式：利用文件代号去查找和存取相应的文件。

11.1 磁盘的记录方式

硬盘是计算机系统中不可缺少的外存设备，软盘是用户最灵活的记录设备。

11.1.1 磁盘记录信息的地址

1. 磁道和扇面
  (1). 硬盘的多个盘片固定在同一根主轴上，称为盘组。
  (2). 无论哪一种盘，信息都记录在盘面的一个个同心圆上，这些同心圆称为磁道。
  (3). 每个磁道被格式化为 512 字节的扇面(扇区)。
2. 柱面：是由每个盘面上统一编号的磁道组成的。
3. 磁盘控制器：是解释来自主机的命令，并向磁盘驱动器发出各种控制信号；同时还要监测驱动器的状态，按规定的数据格式向驱动器读写数据。
4. 簇：将几个扇区组成一组称为簇。系统将簇作为磁盘存储空间的单位，簇的大小总是 2^n 个扇区。
5. 磁盘容量：视磁盘而定。

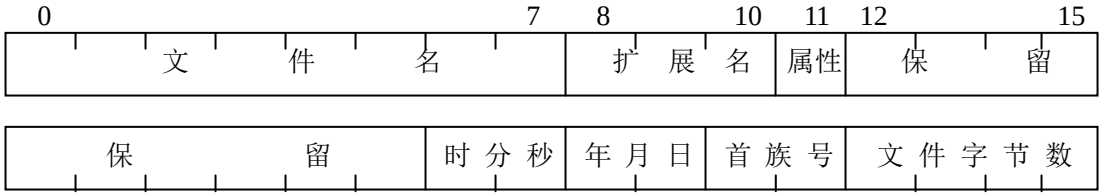
11.1.2 磁盘系统区和数据区

1. 系统区：磁盘的第一个区域是系统区，从 0 面、0 磁道、1 扇区开始。系统区一般用来存放系统存储和维护的信息。系统区由三部分组成：
  (1). 引导记录(帮助系统将操作系统从磁盘装入内存)；
  (2). 文件分配表 FAT(为文件分配磁盘空间)；
  (3). 目录(包括文件名、磁盘地址和文件的状态)。
2. 数据区：用来保存用户文件(系统盘先保存系统文件)。

11.1.3 磁盘目录及文件分配表

1. 磁盘目录
  (1). 每个磁盘上都有一个根目录，它是寻找磁盘文件最重要的一张表。
  (2). 目录中的每一个目录名和文件名都限定在它上层的目录下，这称为路径。

(3). 对每一个文件，DOS 都建立一个 32 字节的目录项，以记录文件名、建立文件的日期、文件的大小以及文件起始簇的地址。格式如下：



ASCII 码为 0DDH

2. 文件分配表 FAT(File Allocation Table) ：用来为文件分配磁盘空间。

(1). FAT 的结构(软盘)：FAT 中每项长 12 位(即 1.5 字节，用 3 位十六进制数表示)。从 0000 项开始。如右图所示：

1). 表头占两项(即 3 个字节)，其中第 2、3 字节固定为 FFFFH，它作为 FAT 的标志，第 1 字节是磁盘规格的说明，具体含义如下：

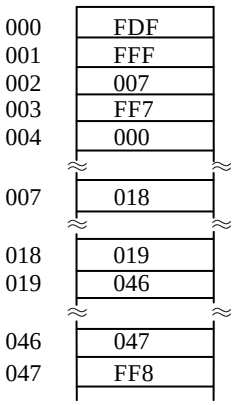
a. FFH—双面盘，8 扇区/道；

b. FEH—单面盘，8 扇区/道；

c. FDH—双面盘，9 扇区/道；

d. FCH—单面盘，9 扇区/道；

e. F8H—硬盘。



2). FAT 中自第 3 项开始用来表示盘上各簇的使用情况，每项对应一簇。

3). FAT 中每项信息的含义如下：

a. 000—此簇未用；

b. FF8~FFFH—表示这是某个文件的最后一簇；

c. FF0~FF7H—此簇为坏簇，不能使用。

d. xxx(其他十六进制数)—文件下一簇的簇号。

(2). 每个文件在 FAT 中的位置的首项序号由目录项的第 26~27 字节的首簇号指出。然后由 FAT 中的该文件的若干项内容(即簇号)指出文件存放的位置，直至簇号为 FF8~FFFH 为止。实际上 FAT 就是一个链表结构，有两个好处：

1). 可以用链式结构分配磁盘空间，这样，每个文件目录项中只需要记录一个文件的首簇号，使目录项显得简明。

2). 有了链表结构以后，文件长度仅仅受磁盘容量的限制，实际上基本不受限制。

11.2 文件代号式磁盘存取

1. 文件代号式存取方式(file handles access)：用文件代号去查找相应的文件，通过读写指针的移动进行随机存取的方式。以字节为单位进行存取。

2. 文件代号：一个完整的文件路径名被送入操作系统后，就被赋予一个 16 位数的简单的文件代号，以后就用该文件代号去查找该文件。

3. 读写指针：DOS 为每个打开的文件管理一个读写指针，该指针总是指向下一次要存取的文件中的字节，这个读写指针可以移动到文件的任意位置，从而能满足随机存取的要求。

4. 该方式对各种文件读写错误采取统一管理。无错时 CF=0，出错时 CF=1 并在 AX 中回送统一的错误代码。

5. DOS 2.0 为文件代号式存取方式提供的 DOS 系统功能调用：INT 21H 的 AH=39H～57H 功能，常用的有 AH=3CH～43H 功能。(详细内容参见书中的 404 页的表 11.3 及附录的 DOS 功能调用。)

11.2.1 路径名和 ASCIZ 串

1. 路径名：路径名说明文件的位置，包括磁盘驱动器号、目录路径和文件名。

2. ASCIZ 串：在路径名的后面加上一个全 0 的字节组成。

11.2.2 文件代号和错误返回代码

1. 文件代号：由打开文件功能或建立文件功能传送到 AX 的一个 16 位数。此代号是从 6 开始顺序排列

的。

2. 标准设备文件代号：不必打开，是由 DOS 预先定义的。

(1). 0 号=标准输入设备

(2). 1 号=标准输出设备

(3). 2 号=标准错误输出设备

(4). 3 号=标准辅助设备

(5). 4 号=标准打印设备

3. 错误返回代码

(1). CF=0：建立或打开文件成功，文件代号送 AX。如果是文件读写，则也是成功的。

(2). CF=1：文件操作不成功，AX 中包含错误代码。错误代码表见书 406 页表 11.4 所示。

文件的属性字节

D7	D6	D5	D4	D3	D2	D1	D0
		归档位	子目录	卷标	系统文件	隐含文件	只读文件

11.2.3 文件属性：一个说明文件特性的字节。

1. 属性字节保存在文件目录项中的 0BH 字节，其各位的含义如右：

2. AH=43H 的 INT 21H 功能调用：检验和改变文件属性，AL=0 检验文件属性，AL=1 改变文件属性，CX=新属性，DS:DX=ASCIZ 串的首地址。返回 CF=0 操作成功，AL=0，CX=属性；CF=1 操作失败，AX=错误代码。

11.2.4 写磁盘文件

1. AH=3CH：建立文件(写新文件)，DS:DX=ASCIZ 串的首地址，CX=文件属性。返回 CF=0 操作成功，AX=文件代号；CF=1 操作失败，AX=错误代码。

2. AH=3DH：打开文件(对已有文件进行读写)，DS:DX=ASCIZ 串的首地址，AL=存取代码。返回 CF=0 操作成功，AX=文件代号；CF=1 操作失败，AX=错误代码。其中存取代码为：

(1). 0——为读而打开文件，

(2). 1——为写而打开文件，

(3). 2——为读和写而打开文件。

3. AH=40H：写文件或标准设备，DS:DX=数据缓冲区首地址，BX=文件代号，CX=写入的字节数。返回 CF=0 操作成功，AX=实际写入的字节数；CF=1 操作错误，AX=错误代码。

4. AH=3EH：关闭文件，BX=文件代号。返回 CF=0 操作成功；CF=1 操作失败，AX=错误代码。

11.2.5 读磁盘文件

1. 打开文件取得文件代号。

2. AH=3FH：读文件或标准设备，DS:DX=数据缓冲区首地址，BX=文件代号，CX=读取的字节数。返回 CF=0 操作成功，AX=实际读取的字节数；CF=1 操作错误，AX=错误代码。

3. 关闭文件。

11.2.6 移动读写指针

1. 读写指针：操作系统对文件保存的确定应从文件的什么地方读出或应往什么地方写入的指针。

2. AH=42H：移动读写指针，CX=所需字节的偏移地址(高位字)，DX=所需字节的偏移地址(低位字)，AL=方式码，BX=文件代号。返回 CF=0 操作成功，DX:AX=新指针位置；CF=1 操作失败，AX=错误代码。

(1). 方式 0(AL=0)：绝对移动方式。偏移值从文件首开始计算。

(2). 方式 1(AL=1)：相对移动方式。当前的指针值加上偏移值作为新的指针值(可找出当前指针位置)。

(3). 方式 2(AL=2)：绝对倒移方式。新的指针位置通过把偏移值和文件尾的位置相加而确定(可找出文件长度)。

11.3 字符设备的文件代号式 I/O

1. 当一个用户得到控制权后，它就得到了五个已打开的文件代号。这五个文件代号是：

(1). 0000：输入设备，通常是键盘。输入能转向。

(2). 0001：输出设备，通常是显示器。输出能转向。

(3). 0002：错误输出设备，总是显示器。输出不能转向。
(4). 0003：辅助设备，一般为通信设备。
(5). 0004：标准打印机(0 号打印机)。

2. 对 LPT1、LPT2 等可用功能 3DH 打开一个指定的设备文件，取得其代号。
3. 对 COM2 等也可用功能 3DH 打开一个指定的设备文件，取得其代号。

11.4 BIOS 磁盘存取功能

1. BIOS 磁盘操作 INT 13H 处理的记录都是一个扇区的大小，都是以物理的磁道号和扇区号来寻址的。
2. 读、写和检验磁盘文件之前，初始化下列寄存器：
(1). AH=要执行的操作：读、写、检验或格式化。
(2). AL=扇区数。
(3). CH=磁道号。
(4). CL=起始的扇区号。
(5). DH=磁头号，对软盘是 0 或 1。
(6). DL=驱动器号。软盘：0=驱动器 A，1=驱动器 B，2=驱动器 C 等。
硬盘：80H=驱动器 1，81H=驱动器 2，…
(7). ES:BX=数据区中 I/O 缓冲区的地址(除检验操作外)。

11.4.1 BIOS 磁盘操作：BIOS 的 INT 13H 要求在 AH 寄存器中指定操作。

1. AH=00：复位软盘系统。对软盘控制器硬件复位。
2. AH=01：读取磁盘状态。返回 AL=状态字节。
3. AH=02：读磁道。返回：读成功 AH=0，AL=读取的扇区数；读失败 AH=出错代码。
4. AH=03：写磁道。返回：写成功 AH=0，AL=写入的扇区数；写失败 AH=出错代码。
5. AH=04：检验磁盘扇区。返回：成功 AH=0，AL=检验的扇区数；失败 AH=出错代码。
6. AH=05:格式化盘磁道,ES:BX=磁道地址区。返回:成功 AH=0；失败 AH=出错代码。其中 ES:BX 指向一组磁道地址区，对磁道上的每一个扇区，必须有一个格式为 T/H/S/B 的四字节的数据项。即：
(1). T=磁道号。
(2). H=磁头号。
(3). S=扇区号。
(4). B= 每扇区的字节数(00=128B， 01=256B， 02=512B，03=1024B)。

AH	原　　因
01	给磁盘 I/O 传送了非法命令
02	磁盘上没有发现地址标记
03	试图往写保护盘上写
04	没有找到指定的扇区
08	DMA 超载运行
09	DMA 超过 64K 的限制
10	读盘数据错(CRC)
20	软盘控制器出错
40	随机移动失败
80	回答失败

11.4.2 状态字节：对上述 AH=02、03、04、05 的磁盘操作，如果操作成功，则 CF 和 AH 置为 0；如操作失败，CF=1，AH 中返回表示出错原因的状态代码(如右所示)。

11.4.3 BIOS 磁盘操作举例：BIOREAD 程序。

附录：《IBM—PC 汇编语言程序设计》习题参考答案

第一章. 习 题

1.1 用降幂法和除法将下列十进制数转换为二进制数和十六进制数：

(1) 369 (2) 10000 (3) 4095 (4) 32767

答：(1) 369=1 0111 0001B=171H

(2) 10000=10 0111 0001 0000B=2710H

(3) 4095=1111 1111 1111B=FFFH

(4) 32767=111 1111 1111 1111B=7FFFH

1.2 将下列二进制数转换为十六进制数和十进制数：

(1) 10 1101 (2) 1000 0000 (3) 1111 1111 1111 1111 (4) 1111 1111

答：(1) 10 1101B=2DH=45

(2) 1000 0000B=80H=128

(3) 1111 1111 1111 1111B=FFFFH=65535

(4) 1111 1111B=FFH=255

1.3 将下列十六进制数转换为二进制数和十进制数：

(1) FA (2) 5B (3) FFFE (4) 1234

答：(1) FAH=1111 1010B=250

(2) 5BH=101 1011B=91

(3) FFFEH=1111 1111 1111 1110B=65534

(4) 1234H=1 0010 0011 0100B=4660

1.4 完成下列十六进制数的运算，并转换为十进制数进行校核：

(1) 3A+B7 (2) 1234+AF (3) ABCD-FE (4) 7AB×6F

答：(1) 3A+B7H=F1H=241

(2) 1234+AFH=12E3H=4835

(3) ABCD-FEH=AACFH=43727

(4) 7AB×6FH=35325H=217893

1.5 下列各数均为十进制数，请用 8 位二进制补码计算下列各题，并用十六进制数表示其运算结果。

(1) (-85)+76 (2) 85+(-76) (3) 85-76 (4) 85-(-76) (5) (-85)-76 (6) -85-(-76)

答：(1) (-85)+76=1010 1011B+0100 1100B=1111 0111B=0F7H；CF=0；OF=0

(2) 85+(-76)=0101 0101B+1011 0100B=0000 1001B=09H；CF=1；OF=0

(3) 85-76=0101 0101B-0100 1100B=0101 0101B+1011 0100B=0000 1001B=09H；CF=0；OF=0

(4) 85-(-76)=0101 0101B-1011 0100B=0101 0101B+0100 1100B=10100001B=0A1H；CF=0；OF=1

(5) (-85)-76=1010 1011B-0100 1100B=1010 1011B+1011 0100B=0101 1111B=5FH；CF=0；OF=1

(6) -85-(-76)=1010 1011B-1011 0100B=1010 1011B+0100 1100B=11110111B=0F7H；CF=0；OF=0

1.6 下列各数为十六进制表示的 8 位二进制数，请说明当它们分别被看作是用补码表示的带符号数或无符号数时，它们所表示的十进制数是什么？

(1) D8 (2) FF

答：(1) D8H 表示的带符号数为 -40，D8H 表示的无符号数为 216；

(2) FFH 表示的带符号数为 -1， FFH 表示的无符号数为 255。

1.7 下列各数均为用十六进制表示的 8 位二进制数，请说明当它们分别被看作是用补码表示的数或字符的 ASCII 码时，它们所表示的十进制数及字符是什么？

(1) 4F (2) 2B (3) 73 (4) 59

答：(1) 4FH 表示的十进制数为 79，4FH 表示的字符为 O；

(2) 2BH 表示的十进制数为 43，2BH 表示的字符为 +；

(3) 73H 表示的十进制数为 115，73H 表示的字符为 s；

(4) 59H 表示的十进制数为 89，59H 表示的字符为 Y。

1.8 请写出下列字符串的 ASCII 码值。

For example,

This is a number 3692.

答：46H 6FH 72H 20H 65H 78H 61H 6DH 70H 6CH 65H 2CH 0AH 0DH

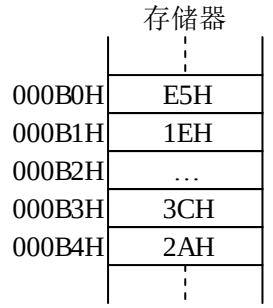
54H 68H 69H 73H 20H 69H 73H 20H 61H 20H 6EH 75H 6DH 62H 65H

72H 20H 33H 36H 39H 32H 2EH 0AH 0DH

第二章. 习 题

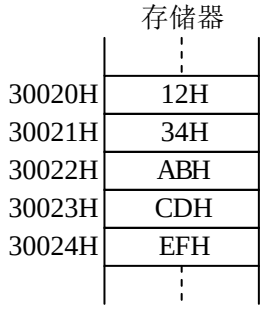
2.1 在 80x86 微机的输入/输出指令中，I/O 端口号通常是由 DX 寄存器提供的，但有时也可以在指令中直接指定 00~FFH 的端口号。试问可直接由指令指定的 I/O 端口数。
答：可直接由指令指定的 I/O 端口数为 256 个。

2.2 有两个 16 位字 1EE5H 和 2A3CH 分别存放在 80x86 微机的存储器的 000B0H 和 000B3H 单元中，请用图表示出它们在存储器里的存放情况。
答：存储器里的存放情况如右上图所示。



2.2 题的信息存放情况

2.3 在 IBM PC 机的存储器中存放信息如右下图所示。试读出 30022H 和 30024H 字节单元的内容，以及 30021H 和 30022H 字单元的内容。
答：30022H 字节单元的内容为 ABH；30024H 字节单元的内容为 EFH。
30021H字单元的内容为 AB34H; 30022H 字单元的内容为CDABH。



2.3 题的信息存放情况

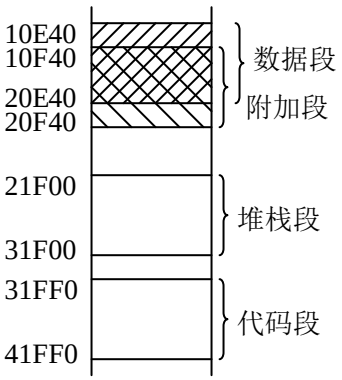
2.4 在实模式下，段地址和偏移地址为 3017:000A 的存储单元的物理地址是什么？如果段地址和偏移地址是 3015:002A 和 3010:007A 呢？
答：3017:000A、3015:002A 和 3010:007A 的存储单元的物理地址都是 3017AH。

2.5 如果在一个程序开始执行以前(CS)=0A7F0H，(如 16 进制数的最高位为字母，则应在其前加一个 0) (IP)=2B40H，试问该程序的第一个字的物理地址是多少？
答：该程序的第一个字的物理地址是 0AAA40H。

2.6 在实模式下，存储器中每一段最多可有 10000H 个字节。如果用调试程序 DEBUG 的 r 命令在终端上显示出当前各寄存器的内容如下，请画出此时存储器分段的示意图，以及条件标志 OF、SF、ZF、CF 的值。

```
C>debug
-r
AX=0000  BX=0000  CX=0079  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000  DS=10E4  ES=10F4  SS=21F0  CS=31FF
IP=0100  NV  UP  DI  PL  NZ  NA  PO  NC
```

答：此时存储器分段的示意图如右图所示。OF、SF、ZF、CF 的值都为 0。



2.6 题的存储器分段示意图

2.7 下列操作可使用那些寄存器？

(1) 加法和减法	数据寄存器等
(2) 循环计数	CX
(3) 乘法和除法	AX、DX，乘数和除数用其他寄存器或存储器
(4) 保存段地址	段寄存器
(5) 表示运算结果为 0	ZF=1
(6) 将要执行的指令地址	CS:IP
(7) 将要从堆栈取出数据的地址	SS:SP

答：答案见题目的右边。

2.8 那些寄存器可以用来指示存储器地址？
答：BX、BP、SI、DI、堆栈操作时的 SP、对应的段地址、386 及其后继机型的 Exx。

2.9 请将下列左边的项和右边的解释联系起来(把所选字母放在括号中)：

(1)	CPU	(M)	A.保存当前栈顶地址的寄存器。
(2)	存储器	(C)	B.指示下一条要执行的指令的地址。
(3)	堆栈	(D)	C.存储程序、数据等信息的记忆装置，微机有 RAM 和 ROM 两种。
(4)	IP	(B)	D.以后进先出方式工作的存储空间。
(5)	SP	(A)	E.把汇编语言程序翻译成机器语言程序的系统程序。
(6)	状态标志	(L)	F.唯一代表存储空间中每个字节单元的地址。
(7)	控制标志	(K)	G.能被计算机直接识别的语言。

(8) 段寄存器 (J) H.用指令的助记符、符号地址、标号等符号书写程序的语言。
(9) 物理地址 (F) I.把若干个模块连接起来成为可执行文件的系统程序。
(10) 汇编语言 (H) J.保存各逻辑段的起始地址的寄存器，8086/8088 机有四个：CS、DS、SS、ES。
(11) 机器语言 (G) K.控制操作的标志，如 DF 位。
(12) 汇编程序 (E) L.记录指令操作结果的标志，共 6 位：OF、SF、ZF、AF、PF、CF。
(13) 连接程序 (I) M.分析、控制并执行指令的部件，由算术逻辑部件 ALU 和寄存器等组成。
(14) 指令 (O) N.由汇编程序在汇编过程中执行的指令。
(15) 伪指令 (N) O.告诉 CPU 要执行的操作(一般还要指出操作数地址)，在程序运行时执行。

答：答案见题目的括号中。

第三章. 习 题

3.1 给定(BX)=637DH，(SI)=2A9BH，位移量 D=7237H，试确定在以下各种寻址方式下的有效地址是什么？
(1) 立即寻址
(2) 直接寻址
(3) 使用 BX 的寄存器寻址
(4) 使用 BX 的简接寻址
(5) 使用 BX 的寄存器相对寻址
(6) 基址变址寻址
(7) 相对基址变址寻址
答：(1) 操作数在指令中，即立即数；
(2) EA=D=7237H；
(3) 无 EA，操作数为(BX)=637DH；
(4) EA=(BX)=637DH；
(5) EA=(BX)+D=0D5B4H；
(6) EA=(BX)+(SI)=8E18H；
(7) EA=(BX)+(SI)+D=1004FH；超过了段的边界，最高进位位丢失，因此 EA=004FH。

3.2 试根据以下要求写出相应的汇编语言指令
(1) 把 BX 寄存器和 DX 寄存器的内容相加，结果存入 DX 寄存器中。
(2) 用寄存器 BX 和 SI 的基址变址寻址方式把存储器中的一个字节与 AL 寄存器的内容相加，并把结果送到 AL 寄存器中。
(3) 用寄存器 BX 和位移量 0B2H 的寄存器相对寻址方式把存储器中的一个字和(CX)相加，并把结果送回存储器中。
(4) 用位移量为 0524H 的直接寻址方式把存储器中的一个字与数 2A59H 相加，并把结果送回存储单元中。
(5) 把数 0B5H 与(AL)相加，并把结果送回 AL 中。
答：(1) ADD DX, BX
(2) ADD AL, [BX][SI]
(3) ADD [BX+0B2H], CX
(4) ADD WORD PTR [0524H], 2A59H
(5) ADD AL, 0B5H

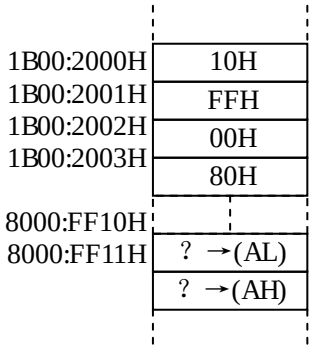
3.3 写出把首地址为 BLOCK 的字数组的第 6 个字送到 DX 寄存器的指令。要求使用以下几种寻址方式：
(1) 寄存器间接寻址
(2) 寄存器相对寻址
(3) 基址变址寻址
答：(1) MOV BX, OFFSET BLOCK
ADD BX, (6－1)*2

```
        MOV   DX, [BX]
(2) MOV   BX, OFFSET  BLOCK          改为：  MOV   BX, (6-1)*2
    MOV   DX, [BX+(6－1)*2]          也可        MOV   DX, BLOCK[BX]
(3) MOV   BX, OFFSET  BLOCK
    MOV   SI, (6－1)*2
    MOV   DX, [BX][SI]
```

3.4 现有(DS)=2000H，(BX)=0100H，(SI)=0002H，(20100H)=12H，(20101H)=34H，(20102H)=56H，(20103H)=78H，(21200H)=2AH，(21201H)=4CH，(21202H)=B7H，(21203H)=65H，试说明下列各条指令执行完后 AX 寄存器的内容。

```
(1) MOV   AX, 1200H
(2) MOV   AX, BX
(3) MOV   AX, [1200H]
(4) MOV   AX, [BX]
(5) MOV   AX, 1100[BX]
(6) MOV   AX, [BX][SI]
(7) MOV   AX, 1100[BX][SI]
```

答：(1) (AX)=1200H
(2) (AX)=0100H
(3) (AX)=4C2AH
(4) (AX)=3412H
(5) (AX)=4C2AH
(6) (AX)=7856H
(7) (AX)=65B7H



3.6 题的作图表示

3.5 给定(IP)=2BC0H，(CS)=0200H，位移量 D=5119H，(BX)=1200H，(DS)=212AH，(224A0H)=0600H，(275B9H)=098AH，试为以下的转移指令找出转移的偏移地址。

(1) 段内直接寻址
(2) 使用 BX 及寄存器间接寻址方式的段内间接寻址
(3) 使用 BX 及寄存器相对寻址方式的段内间接寻址

答：(1) JMP NEAR PTR 5119H ；(IP)=5119H+((IP)+03H)=7CDCH，物理地址 PA=09CDCH
(IP)+03H 是 JMP NEAR PTR 5119H 指令的下一条指令的首地址。

```
(2) JMP   WORD PTR [BX]      ；(IP)=((DS)*10H+(BX))=0600H，PA=02600H
(3) JMP   D[BX]              ；(IP)=((DS)*10H+(BX)+D)=098AH，PA=0298AH
```

3.6 设当前数据段寄存器的内容为 1B00H，在数据段的偏移地址 2000H 单元内，含有一个内容为 0FF10H 和 8000H 的指针，它们是一个 16 位变量的偏移地址和段地址，试写出把该变量装入 AX 的指令序列，并画图表示出来。

答：
```
MOV   BX, [2000H]            ；图示如上所示。
MOV   AX, [2000H+2]
MOV   ES, AX
MOV   AX, ES:[BX]
```

```
MOV   BX, 2000H
LES   BX, [BX]
MOV   AX, ES :[BX]
```

3.7 在 0624H 单元内有一条二字节 JMP SHORT OBJ 指令，如其中位移量为(1) 27H，(2) 6BH，(3) 0C6H，试问转向地址 OBJ 的值是多少？

答：(1) OBJ=0624H+02H+27H=064DH
(2) OBJ=0624H+02H+6BH=0691H
(3) OBJ=0624H+02H+0C6H=05ECH ；C6H 对应的负数为-3AH（向上转移，负位移量）

3.8 假定(DS)=2000H，(ES)=2100H，(SS)=1500H，(SI)=00A0H，(BX)=0100H，(BP)=0010H，数据段中变量名 VAL 的偏移地址为 0050H，试指出下列源操作数字段的寻址方式是什么？其物理地址值是多少？

```
(1) MOV   AX, 0ABH           (2) MOV   AX, BX
(3) MOV   AX, [100H]         (4) MOV   AX, VAL
(5) MOV   AX, [BX]           (6) MOV   AX, ES:[BX]
(7) MOV   AX, [BP]           (8) MOV   AX, [SI]
(9) MOV   AX, [BX+10]        (10) MOV  AX, VAL[BX]
(11) MOV  AX, [BX][SI]       (12) MOV  AX, VAL[BX][SI]
```

答：(1) 立即方式；操作数在本条指令中

(2) 寄存器寻址方式； 操作数为 (BX)=0100H
(3) 直接寻址方式； PA=20100H
(4) 直接寻址方式； PA=20050H
(5) BX 寄存器间接寻址方式； PA=20100H
(6) 附加段 BX 寄存器间接寻址方式； PA=21100H
(7) BP 寄存器间接寻址方式； PA=15010H
(8) SI 寄存器间接寻址方式； PA=200A0H
(9) BX 寄存器相对寻址方式； PA=20110H
(10) BX 寄存器相对寻址方式； PA=20150H
(11) BX 和 SI 寄存器基址变址寻址方式； PA=201A0H
(12) BX 和 SI 寄存器相对基址变址寻址方式； PA=201F0H

3.9 在 ARRAY 数组中依次存储了七个字数据，紧接着是名为 ZERO 的字单元，表示如下：

```
ARRAY   DW   23, 36, 2, 100, 32000, 54, 0
ZERO    DW   ?
```

(1) 如果 BX 包含数组 ARRAY 的初始地址，请编写指令将数据 0 传送给 ZERO 单元。
(2) 如果 BX 包含数据 0 在数组中的位移量，请编写指令将数据 0 传送给 ZERO 单元。
答：(1)

```
MOV   AX, [BX+(7-1)*2]
MOV   [BX+(7)*2], AX
```

(2)

```
MOV   AX, ARRAY [BX]
MOV   ARRAY [BX+2], AX
```

3.10 如 TABLE 为数据段中 0032 单元的符号名，其中存放的内容为 1234H，试问以下两条指令有什么区别？指令执行完后 AX 寄存器的内容是什么？

```
MOV     AX, TABLE
LEA     AX, TABLE
```

答：MOV AX, TABLE 是将 TABLE 单元的内容送到 AX，(AX)=1234H
LEA AX,TABLE 是将 TABLE 单元的有效地址送到 AX，(AX)=0032H

TABLE	0AH
	00H
	14H
TABLE+3	00H
	1EH
	00H
	28H
	00H
	32H
	00H

3.11 题的 TABLE 存储方式

3.11 执行下列指令后 AX 寄存器中的内容是什么？

```
TABLE   DW   10, 20, 30, 40, 50      ; 000AH, 0014H, 001EH, 0028H, 0032H
ENTRY   DW   3
          ⋮
        MOV     BX, OFFSET   TABLE
        ADD     BX, ENTRY
        MOV     AX, [BX]
```

答：(AX)=1E00H (TABLE 的存储方式如右图所示)

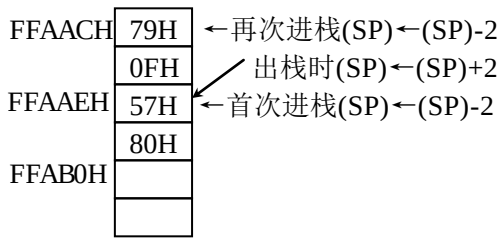
3.12 下列 ASCII 码串(包括空格符)依次存储在起始地址为 CSTRING 的字节单元中：

```
CSTRING   DB   'BASED ADDRESSING'
```

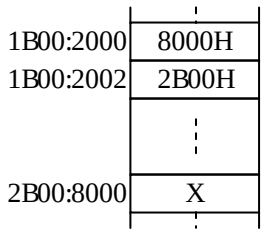
请编写指令将字符串中的第 1 个和第 7 个字符传送给 DX 寄存器。
答：

```
MOV   DH, CSTRING
MOV   DL, CSTRING+7-1
```

3.13 已知堆栈段寄存器 SS 的内容是 0FFA0H，堆栈指针寄存器 SP 的内容是 00B0H，先执行两条把 8057H 和 0F79H 分别进栈的 PUSH 指令，再执行一条 POP 指令。试画出堆栈区和 SP 的内容变化过程示意图(标出存储单元的物理地址)。
答：堆栈区和 SP 的内容变化过程示意图如下左图所示。



3.13 题的堆栈区和 SP 的内容变化过程示意图



3.14 题的存储区情况

3.14 设(DS)=1B00H，(ES)=2B00H，有关存储单元的内容如上右图所示。请写出两条指令把字变量 X

装入 AX 寄存器。

答：MOV BX, [2000H]
MOV AX, ES:[BX]

3.15 求出以下各十六进制数与十六进制数 62A0H 之和，并根据结果设置标志位 SF、ZF、CF 和 OF 的值。

(1) 1234H (2) 4321H (3) CFA0H (4) 9D60H

答：(1) 和为 74D4H；SF=0，ZF=0，CF=0，OF=0
(2) 和为 A5C1H；SF=1，ZF=0，CF=0，OF=1
(3) 和为 3240H；SF=0，ZF=0，CF=1，OF=0
(4) 和为 0000H；SF=0，ZF=1，CF=1，OF=0

3.16 求出以下各十六进制数与十六进制数 4AE0H 的差值，并根据结果设置标志位 SF、ZF、CF 和 OF 的值。

(1) 1234H (2) 5D90H (3) 9090H (4) EA04H

答：(1) 差为 C754H；SF=1，ZF=0，CF=1，OF=0
(2) 差为 12B0H；SF=0，ZF=0，CF=0，OF=0
(3) 差为 45B0H；SF=0，ZF=0，CF=0，OF=1
(4) 差为 9F24H；SF=1，ZF=0，CF=0，OF=0

3.17 写出执行以下计算的指令序列，其中 X、Y、Z、R、W 均为存放 16 位带符号数单元的地址。

(1) Z←W+(Z-X) (2) Z←W-(X+6)-(R+9)
(3) Z←(W*X)/(Y+6)，R←余数 (4) Z←((W-X)/5*Y)*2

答：(1)
```
MOV   AX, Z      ；以下程序都未考虑带符号数的溢出
SUB   AX, X
ADD   AX, W
MOV       Z, AX
```
(2)
```
MOV   BX, X
ADD   BX, 6
MOV   CX, R
ADD   CR, 9
MOV       AX, W
SUB   AX, BX
SUB   AX, CX
MOV       Z, AX
```
(3)
```
ADD   Y, 6
MOV       AX, W
IMUL   X
IDIV   Y
MOV       Z, AX
MOV       R, DX
```
(4)
```
MOV   AX, W
SUB   AX, X
CWD
MOV   BX, 5
IDIV   BX
IMUL   Y
SHL   AX, 1   ；((DX),(AX))*2
RCL   DX, 1
```

3.18 已知程序段如下：

```
MOV     AX, 1234H    ；(AX)=1234H，标志位不变
MOV     CL, 4        ；(AX)和标志位都不变
ROL         AX, CL       ；(AX)=2341H，CF=1，SF 和 ZF 不变
DEC         AX           ；(AX)=2340H，CF=1 不变，SF=0，ZF=0
MOV     CX, 4        ；(AX)和标志位都不变
MUL     CX           ；(AX)=8D00H，CF=OF=0，其它标志无定义
INT     20H
```

试问：

(1) 每条指令执行完后，AX 寄存器的内容是什么？

(2) 每条指令执行完后，进位、符号和零标志的值是什么？

(3) 程序结束时，AX 和 DX 的内容是什么？

答：(1) 见注释；

(2) 见注释；

(3) (AX)=8D00H，(DX)=0

3.19 下列程序段中的每条指令执行完后，AX 寄存器及 CF、SF、ZF 和 OF 的内容是什么？

```
MOV     AX, 0           ；(AX)=0,          标志位不变
DEC       AX              ；(AX)=0FFFFH,  CF 不变，SF=1，ZF=0，OF=0
ADD     AX, 7FFFH       ；(AX)=7FFEH,    CF=1，SF=0，ZF=0，OF=0
ADD     AX, 2           ；(AX)=8000H,    CF=0，SF=1，ZF=0，OF=1
NOT     AX              ；(AX)=7FFFH,    标志位不变
SUB     AX, 0FFFFH      ；(AX)=8000H,    CF=1，SF=1，ZF=0，OF=1
ADD     AX, 8000H       ；(AX)=0，       CF=1，SF=0，ZF=1，OF=1
SUB     AX, 1           ；(AX)=0FFFFH,  CF=1，SF=1，ZF=0，OF=0
AND     AX, 58D1H       ；(AX)=58D1H,    CF=0，SF=0，ZF=0，OF=0
SAL     AX, 1           ；(AX)=0B1A2H,  CF=0，SF=1，ZF=0，OF=1
SAR     AX, 1           ；(AX)=0D8D1H,  CF=0，SF=1，ZF=0，OF=0
NEG     AX              ；(AX)= 272FH,   CF=1，SF=0，ZF=0，OF=0
ROR     AX, 1           ；(AX)= 9397H,   CF=1，SF 和 ZF 不变，OF=1
```

答：见注释。

3.20 变量 DATAX 和变量 DATAY 的定义如下：

```
DATAX   DW    0148H
        DW    2316H
DATAY   DW    0237H
        DW    4052H
```

请按下列要求写出指令序列：

(1) DATAX 和 DATAY 两个字数据相加，和存放在 DATAY 中。

(2) DATAX 和 DATAY 两个双字数据相加，和存放在从 DATAY 开始的双字单元中。

(3) 解释下列指令的作用：

```
STC
MOV         BX, DATAX
ADC     BX, DATAY
```

(4) DATAX 和 DATAY 两个字数据相乘(用 MUL)。

(5) DATAX 和 DATAY 两个双字数据相乘(用 MUL)。

(6) DATAX 除以 23(用 DIV)。

(7) DATAX 双字除以字 DATAY (用 DIV)。

答：(1)

```
MOV    AX, DATAX
ADD    DATAY, AX
MOV    AX, DATAX+2
ADD    DATAY+2, AX
```

(2)

```
MOV    AX, DATAX
ADD    DATAY, AX
MOV    AX, DATAX+2
ADC         DATAY+2, AX
MOV    DATAY+4, 0                    ；用于存放进位位
ADC         DATAY+4, 0
```

(3) DATAX 和 DATAY 两个字数据之和加 1，结果存入 BX 寄存器。

(4)

```
RESULT1  DW    0
         DW    0
RESULT2  DW    0
         DW    0
           ┇
```

```
        MOV   AX, DATAX
        MUL   DATAY
        MOV   RESULT1 , AX
        MOV   RESULT1+2, DX
        MOV   AX, DATAX+2
        MUL   DATAY+2
        MOV   RESULT2 , AX
        MOV   RESULT2+2, DX
(5) AA  DW    0
    BB  DW    0
    CC  DW    0
    DD  DW    0
          ┆
        MOV   AX, DATAX
        MUL   DATAY
        MOV   AA , AX
        MOV   BB, DX
        MOV   AX, DATAX
        MUL   DATAY+2
        ADD   BB, AX
        ADC       CC, DX
        MOV   AX, DATAX+2
        MUL   DATAY
        ADD   BB, AX
        ADC       CC, DX
        ADC       DD, 0
        MOV  AX, DATAX+2
        MUL   DATAY+2
        ADD   CC, AX
        ADC       DD, DX
(6) MOV   AX, DATAX
    MOV   BL, 23
    DIV   BL
(7) MOV   DX, DATAX+2
    MOV   AX, DATAX
    DIV   DATAY
```

3.21 写出对存放在 DX 和 AX 中的双字长数求补的指令序列。

答：
```
NEG   DX
NEG   AX
SBB   DX, 0
```
也可为：
```
      NOT     DX
NOT   AX
ADD   AX, 1
      ADC     DX, 0
```

3.22 试编写一程序求出双字长数的绝对值。双字长数在 A 和 A+2 单元中，结果存放在 B 和 B+2 单元中。

答：程序段如下：
```
          MOV     AX, A
          MOV     DX, A+2
          CMP     DX, 0
          JNS     ZHENSHU    ；不是负数则转走
          NEG     DX
          NEG     AX
          SBB     DX, 0
ZHENSHU:  MOV     B, AX
          MOV     B+2, DX
          INT     20H
```

3.23 假设(BX)=0E3H，变量 VALUE 中存放的内容为 79H，确定下列各条指令单独执行后的结果。

(1) XOR BX, VALUE ；(BX)=9AH，CF、OF 都为 0，AF 无定义，SF=1，ZF=0，PF=1

(2) AND BX, VALUE ；(BX)=61H，CF、OF 都为 0，AF 无定义，SF=0，ZF=0，PF=0

(3) OR BX, VALUE ；(BX)=0FBH，CF、OF 都为 0，AF 无定义，SF=1，ZF=0，PF=0
(4) XOR BX, 0FFH ；(BX)=1CH，CF、OF 都为 0，AF 无定义，SF=0，ZF=0，PF=0
(5) AND BX, 0 ；(BX)=00H，CF、OF 都为 0，AF 无定义，SF=0，ZF=1，PF=1
(6) TEST BX, 01H ；(BX)=0E3H，CF、OF 都为 0，AF 无定义，SF=1，ZF=0，PF=0
答：见注释。

3.24 试写出执行下列指令序列后 BX 寄存器的内容。执行前(BX)=6D16H。

```
MOV     CL, 7
SHR     BX, CL
```

答：(BX)=00DAH。

3.25 试用移位指令把十进制数+53 和-49 分别乘以 2。它们应该用什么指令？得到的结果是什么？如果要除以 2 呢？

答：

```
MOV     AL, 53
SAL     AL, 1           ; (AL)=(+53*2)=6AH
MOV     AL, -49
SAL     AL, 1           ; (AL)=(-49*2)=9EH
MOV     AL, 53
SAR     AL, 1           ; (AL)=(53/2)= 1AH
MOV     AL, -49
SAR     AL, 1           ; (AL)=(-49/2)=0E7H
```

3.26 试分析下面的程序段完成什么功能？

```
MOV     CL, 04
SHL     DX, CL
MOV     BL, AH
SHL     AX, CL
SHR     BL, CL
OR      DL, BL
```

答：本程序段将 ((DX),(AX)) 的双字同时左移 4 位，即将此双字乘以 10H(16)。

3.27 假定(DX)=0B9H，(CL)=3，(CF)=1，确定下列各条指令单独执行后 DX 中的值。

(1) SHR DX, 1 ；(DX)=05CH
(2) SAR DX, CL ；(DX)=17H
(3) SHL DX, CL ；(DX)=5C8H
(4) SHL DL, 1 ；(DX)=72H
(5) ROR DX, CL ；(DX)=2017H
(6) ROL DL, CL ；(DX)=0CDH
(7) SAL DH, 1 ；(DX)=0B9H
(8) RCL DX, CL ；(DX)=2CCH
(4) RCR DL, 1 ；(DX)=0DCH
答：见注释。

3.28 下列程序段执行完后，BX 寄存器的内容是什么？

```
MOV     CL, 3
MOV     BX, 0B7H
ROL     BX,1
ROR     BX, CL
```

答：(BX)=0C02DH。

3.29 假设数据段定义如下：

```
CONAME      DB ‘SPACE  EXPLORERS  INC.’
PRLINE      DB  20 DUP (‘’)
```

用串指令编写程序段分别完成以下功能：
(1) 从左到右把 CONAME 中的字符串传送到 PRLINE。
(2) 从右到左把 CONAME 中的字符串传送到 PRLINE。
(3) 把 CONAME 中的第 3 和第 4 个字节装入 AX。
(4) 把 AX 寄存器的内容存入从 PRLINE+5 开始的字节中。

(5) 检查 CONAME 字符串中有无空格字符，如有则把第一个空格字符的地址传送给 BX 寄存器。

答：
```
(1) MOV     CX, 20
    CLD
    MOV     SI, SEG  CONAME
    MOV     DS, SI
    MOV     ES, SI
    LEA     SI, CONAME
    LEA     DI, PRLINE
    REP     MOVSB
(2) MOV     CX, 20
    STD
    MOV     SI, SEG  CONAME
    MOV     DS, SI
    MOV     ES, SI
    LEA     SI, CONAME
    ADD     SI, 20-1
    LEA     DI, PRLINE
    ADD     DI, 20-1
    REP     MOVSB
(3) MOV     AX, WORD PTR CONAME+3-1
(4) MOV     WORD PTR PRLINE +5, AX
(5) MOV     AL, ‘ ’                ；空格的 ASCII 码送 AL 寄存器
    CLD
    MOV     DI, SEG  CONAME
    MOV     ES, DI
    LEA     DI, CONAME
    REPNE   SCASB
    JNE     NEXT
    DEC     DI
    MOV     BX, DI
NEXT:       ┆
```

3.30 编写程序段，把字符串 STRING 中的‘&’字符用空格符代替。

```
STRING  DB  ‘The date is FEB&03’
```

答：程序段如下：

```
    MOV     CX, 18
    MOV     AL, ‘&’
    CLD
    MOV     DI, SEG  STRING
    MOV     ES, DI
    LEA     DI, STRING
    REPNE   SCASB
    JNE     NEXT
    DEC     DI
    MOV     ES：BYTE  PTR  [DI], ‘ ’  ；送空格符
NEXT:       ┆
```

3.31 假设数据段中数据定义如下：

```
STUDENT_NAME        DB  30  DUP (?)
STUDENT_ADDR    DB  9   DUP (?)
PRINT_LINE          DB  132     DUP (?)
```

分别编写下列程序段：

(1) 用空格符清除 PRINT_LINE 域。

(2) 在 STUDENT_ADDR 中查找第一个‘-’。

(3) 在 STUDENT_ADDR 中查找最后一个‘-’。

(4) 如果 STUDENT_NAME 域中全是空格符时，填入‘*’。

(5) 把 STUDENT_NAME 移到 PRINT_LINE 的前 30 个字节中，把 STUDENT_ ADDR 移到 PRINT_LINE 的后 9 个字节中。

答：公共的程序段如下：

```
    MOV     DI, DS
    MOV     ES, DI
(1) MOV     CX, 132
    MOV     AL., ‘  ’           ；空格的 ASCII 码送 AL 寄存器
    CLD
    LEA     DI, PRINT_LINE
    REP     STOSB
(2) MOV     CX, 9
    MOV     AL., ‘-’
    CLD
    LEA     DI, STUDENT_ ADDR
    REPNE   SCASB
    JNE     NO_DASH
    DEC     DI
NO_DASH:    ┆
(3) MOV     CX, 9
    MOV     AL., ‘-’
    STD
    LEA     DI, STUDENT_ ADDR
    ADD     DI, 9-1
    REPNE   SCASB
    JNE     NO_DASH
    INC     DI
NO_DASH:    ┆
(4) MOV     CX, 30
    MOV     AL, ‘  ’            ；空格的 ASCII 码送 AL 寄存器
    CLD
    LEA     DI, STUDENT_NAME
    REPE    SCASB
    JNE     NEXT
    MOV     CX, 30
    MOV     AL, ‘*’             ；“*”的 ASCII 码送 AL 寄存器
    LEA     DI, STUDENT_NAME
    REP     STOSB
NEXT:       ┆
(5) MOV     CX, 30
    CLD
    LEA     SI, STUDENT_NAME
    LEA     DI, PRINT_LINE
    REP     MOVSB
    MOV     CX, 9
    STD
    LEA     SI, STUDENT_ADDR+9-1
    LEA     DI, PRINT_LINE+132-1
    REP     MOVSB
```

3.32 编写一程序段：比较两个 5 字节的字符串 OLDS 和 NEWS，如果 OLDS 字符串不同于 NEWS 字符串则执行 NEW_LESS；否则顺序执行程序。

答：程序段如下：

```
MOV     CX, 5
CLD
MOV     DI, SEG   OLDS
MOV     DS, DI
MOV     ES, DI
LEA     SI, OLDS
LEA     DI, NEWS
REPE    CMPSB
JNE     NEW_LESS
            ┆
```

NEW_LESS: ┆

3.33 假定 AX 和 BX 中的内容为带符号数，CX 和 DX 中的内容为无符号数，请用比较指令和条件转移指令实现以下判断：

(1) 若 DX 的内容超过 CX 的内容，则转去执行 EXCEED。
(2) 若 BX 的内容大于 AX 的内容，则转去执行 EXCEED。
(3) 若 CX 的内容等于 0，则转去执行 ZERO。
(4) BX 与 AX 的内容相比较是否产生溢出？若溢出则转 OVERFLOW。
(5) 若 BX 的内容小于等于 AX 的内容，则转 EQ_SMA。
(6) 若 DX 的内容低于等于 CX 的内容，则转 EQ_SMA。

答：(1)
```
CMP  DX, CX
JA   EXCEED
```
(2)
```
CMP  BX, AX
JG   EXCEED
```
(3)
```
JCXZ ZERO
```
(4)
```
CMP  BX, AX
JO   OVERFLOW
```
(5)
```
CMP  BX, AX
JLE  EQ_SMA
```
(6)
```
CMP  DX, CX
JBE  EQ_SMA
```

3.34 试分析下列程序段：

```
ADD     AX, BX
JNO     L1
JNC     L2
SUB     AX, BX
JNC     L3
JNO     L4
JMP     SHORT   L5
```

如果 AX 和 BX 的内容给定如下：

	AX	BX
(1)	147BH	80DCH
(2)	B568H	42C8H
(3)	42C8H	608DH
(4)	D023H	9FD0H
(5)	94B7H	B568H

问该程序分别在上面 5 种情况下执行后，程序转向哪里？

答：(1) 转向 L1
(2) 转向 L1
(3) 转向 L2
(4) 转向 L5 ；因为加法指令后 AX 中已经是 6FF3H
(5) 转向 L5 ；因为加法指令后 AX 中已经是 4A14H

3.35 指令 CMP AX, BX 后面跟着一条格式为 J… L1 的条件转移指令，其中…可以是 B、NB、BE、NBE、L、NL、LE、NLE 中的任意一个。如果 AX 和 BX 的内容给定如下：

	AX	BX
(1)	1F52H	1F52H
(2)	88C9H	88C9H
(3)	FF82H	007EH
(4)	58BAH	020EH
(5)	FFC5H	FF8BH
(6)	09A0H	1E97H
(7)	8AEAH	FC29H
(8)	D367H	32A6H

问以上 8 条转移指令中的哪几条将引起转移到 L1？

答：(1) JNB、JBE、JNL、JLE
(2) JNB、JBE、JNL、JLE

(3) JNB、JNBE、JL、JLE
(4) JNB、JNBE、JNL、JNLE
(5) JNB、JNBE、JL、JLE
(6) JB、JBE、JL、JLE
(7) JB、JBE、JNL、JNLE
(8) JNB、JNBE、JL、JLE

3.36 假设 X 和 X+2 单元的内容为双精度数 p，Y 和 Y+2 单元的内容为双精度数 q，(X 和 Y 为低位字)试说明下列程序段做什么工作？

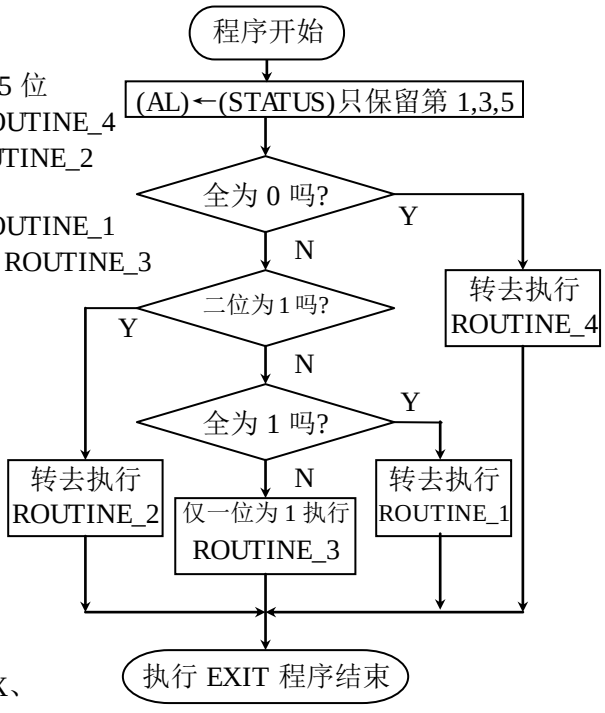
```
        MOV   DX, X+2
        MOV   AX, X
        ADD   AX, X
        ADC   DX, X+2
        CMP   DX, Y+2
        JL    L2
        JG    L1
        CMP   AX, Y
        JBE   L2
L1:     MOV   AX, 1
        JMP   SHORT   EXIT
L2:     MOV   AX, 2
EXIT:   INT   20H
```

答：此程序段判断 p*2＞q，则使(AX)=1 后退出；p*2≤q，则使(AX)=2 后退出。

3.37 要求测试在 STATUS 中的一个字节，如果第 1、3、5 位均为 1 则转移到 ROUTINE_1；如果此三位中有两位为 1 则转移到 ROUTINE_2；如果此三位中只有一位为 1 则转移到 ROUTINE_3；如果此三位全为 0 则转移到 ROUTINE_4。试画出流程图，并编制相应的程序段。

答：程序段如下：

```
        MOV   AL, STATUS
        AND   AL, 00010101B ；只保留第 1、3、5 位
        JZ    ROUTINE_4     ；3 位全为 0 转 ROUTINE_4
        JPE   ROUTINE_2     ；两位为 1 转 ROUTINE_2
        CMP   AL, 00010101B
        JZ    ROUTINE_1     ；3 位全为 1 转 ROUTINE_1
ROUTINE_3:  ┆               ；仅一位为 1 执行 ROUTINE_3
        JMP   EXIT
ROUTINE_1:  ┆
        JMP   EXIT
ROUTINE_2:  ┆
        JMP   EXIT
ROUTINE_4:  ┆
EXIT:   INT   20H
```



3.44 题的程序流程图

3.38 在下列程序的括号中分别填入如下指令：

(1) LOOP L20
(2) LOOPE L20
(3) LOOPNE L20

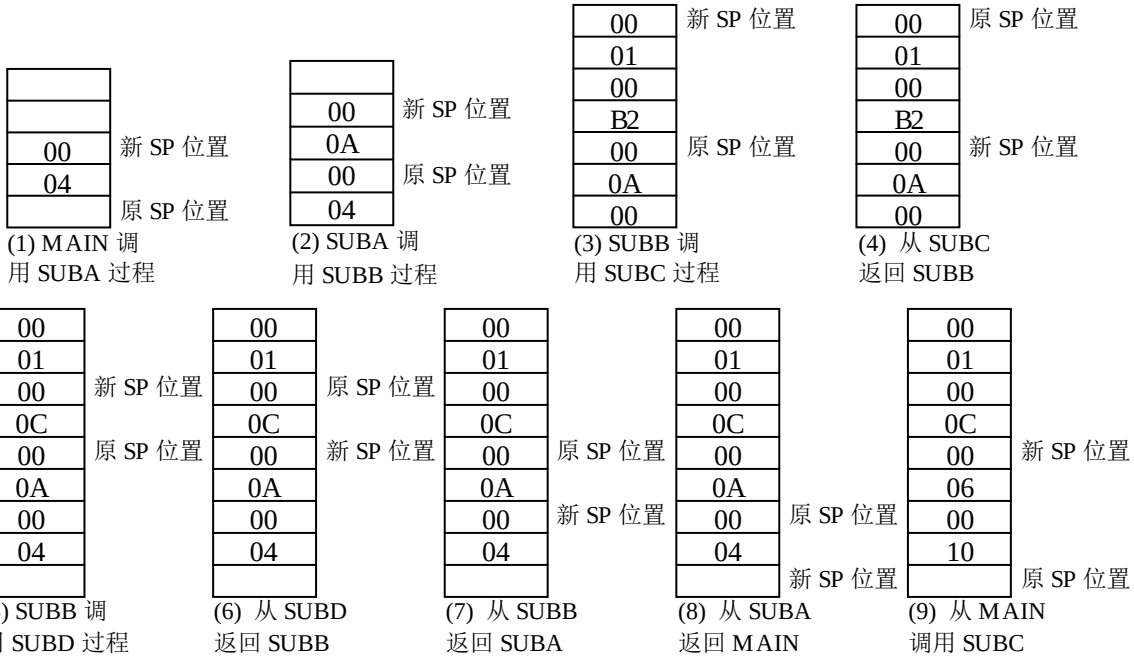
试说明在三种情况下，当程序执行完后，AX、BX、CX、DX 四个寄存器的内容分别是什么？

```
TITLE       EXLOOP.COM
CODESG      SEGMENT
            ASSUME  CS:CODESG, DS: CODSEG, SS: CODSEG
            ORG     100H
BEGIN:      MOV     AX, 01
            MOV     BX, 02
            MOV     DX, 03
            MOV     CX, 04
L20:
```

```
                INC     AX
                ADD     BX, AX
                SHR         DX, 1
                (       )
                RET
CODESG          ENDS
                END   BEGIN
```

答：(1) (AX)=5H，(BX)=10H，(CX)=0H，(DX)=0H
(2) (AX)=2H，(BX)=4H，(CX)=3H，(DX)=1H
(3) (AX)=3H，(BX)=7H，(CX)=2H，(DX)=0H

3.39 考虑以下的调用序列：
(1) MAIN 调用 NEAR 的 SUBA 过程(返回的偏移地址为 0400)；
(2) SUBA 调用 NEAR 的 SUBB 过程(返回的偏移地址为 0A00)；
(3) SUBB 调用 FAR 的 SUBC 过程(返回的段地址为 B200，返回的偏移地址为 0100)；
(4) 从 SUBC 返回 SUBB;
(5) SUBB 调用 NEAR 的 SUBD 过程(返回的偏移地址为 0C00)；
(6) 从 SUBD 返回 SUBB;
(7) 从 SUBB 返回 SUBA;
(8) 从 SUBA 返回 MAIN;
(9) 从 MAIN 调用 SUBC(返回的段地址为 1000，返回的偏移地址为 0600)；
请画出每次调用及返回时的堆栈状态。
答：每次调用及返回时的堆栈状态图如下所示：



3.40 假设(EAX)=00001000H，(EBX)=00002000H，(DS)=0010H，试问下列指令访问内存的物理地址是什么？
(1) MOV ECX，[EAX+EBX]
(2) MOV [EAX+2*EBX]，CL
(3) MOV DH，[EBX+4*EAX+1000H]
答：(1) PA=(DS)*10H+EA=00100H+00001000H+00002000H=00003100H
(2) PA=(DS)*10H+EA=00100H+00001000H+2*00002000H=00005100H
(3) PA=(DS)*10H+EA=00100H+00002000H+4*00001000H+1000H=00007100H

3.41 假设(EAX)=9823F456H，(ECX)=1F23491H，(BX)=348CH，(SI)=2000H，(DI)=4044H。在 DS 段中从偏移地址 4044H 单元开始的 4 个字节单元中，依次存放的内容为 92H，6DH，0A2H 和 4CH，试问下列各条指令执行完后的目的地址及其中的内容是什么？
(1) MOV [SI]，EAX

(2) MOV [BX]，ECX
(3) MOV EBX，[DI]
答：(1) 目的地址为 DS:2000H，内容依次为：56H，0F4H，23H 和 98H
(2) 目的地址为 DS:348CH，内容依次为：91H，34H，0F2H 和 01H
(3) 目的操作数为 EBX 寄存器，(EBX)=4CA26D92H

3.42 说明下列指令的操作

```
(1) PUSH    AX              ；将(AX)压入堆栈
(2) POP     ESI             ；将堆栈中的双字弹出到 ESI 寄存器中
(3) PUSH    [BX]            ；将((BX))对应存储单元中的字压入堆栈
(4) PUSHAD                  ；32 位通用寄存器依次进栈
(5) POP     DS              ；将堆栈中的字弹出到 DS 寄存器中
(6) PUSH    4               ；将立即数 4 以字的方式压入堆栈
```

答：见注释。

3.43 请给出下列各指令序列执行完后目的寄存器的内容。

```
(1) MOV     EAX，299FF94H
    ADD     EAX，34FFFFH        ；(EAX)= 2CEFF93H
(2) MOV     EBX，40000000
    SUB     EBX，1500000        ；(EBX)= 3EB00000H
(3) MOV     EAX，39393834H
    AND     EAX，0F0F0F0FH      ；(EAX)= 09090804H
(4) MOV     EDX，9FE35DH
    XOR     EDX，0F0F0F0H       ；(EDX)= 6F13ADH
```

答：见注释。

3.44 请给出下列各指令序列执行完后目的寄存器的内容。

```
(1) MOV     BX，-12
    MOVSX   EBX，BX             ；(EBX)= 0FFFF FFF4H
(2) MOV     CL，-8
    MOVSX   EDX，CL             ；(EDX)= 0FFFF FFF8H
(3) MOV     AH，7
    MOVZX   ECX，AH             ；(ECX)= 0000 0007H
(4) MOV     AX，99H
    MOVZX   EBX，AX             ；(EBX)= 0000 0099H
```

答：见注释。

3.45 请给出下列指令序列执行完后 EAX 和 EBX 的内容。

```
    MOV     ECX，307 F455H
    BSF     EAX，ECX            ；(EAX)= 0D
    BSR     EBX，ECX            ；(EBX)= 25D
```

答：见注释。

3.46 请给出下列指令序列执行完后 AX 和 DX 的内容。

```
    MOV     BX，98H
    BSF     AX，BX              ；(AX)= 3D
    BSR     DX，BX              ；(DX)= 7D
```

答：见注释。

3.47 请编写一程序段，要求把 ECX、EDX 和 ESI 的内容相加，其和存入 EDI 寄存器中(不考虑溢出)。

```
答：MOV     EDI，0                  也可为：    MOV     EDI，ECX
    ADD     EDI，ECX                            ADD     EDI，EDX
    ADD     EDI，EDX                            ADD     EDI，ESI
    ADD     EDI，ESI
```

3.48 请说明 IMUL BX，DX，100H 指令的操作。
答：(BX)←(DX)*100H

3.49 试编写一程序段，要求把 BL 中的数除以 CL 中的数，并把其商乘以 2，最后的结果存入 DX 寄存器中。

答：
```
MOV     AL，BL
MOV     AH，0                ；假定为无符号数，否则用 CBW 指令即可
DIV     CL
MOV     AH，0
SHL     AX，1
MOV     DX，AX
```

3.50 请说明 JMP DI 和 JMP [DI]指令的区别。

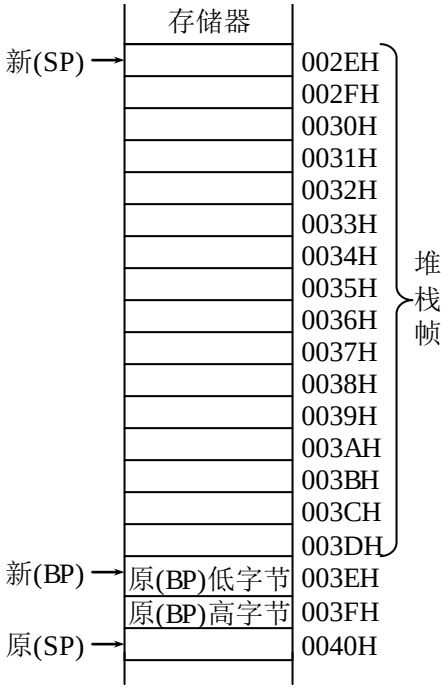
答：JMP DI 是转移到以(DI)内容为偏移地址的单元去执行指令；JMP [DI]是转移到以(DI)间接寻址的内存单元内容为偏移地址的单元去执行指令。

3.51 试编写一程序段，要求在长度为 100H 字节的数组中，找出大于 42H 的无符号数的个数并存入字节单元 UP 中；找出小于 42H 的无符号数的个数并存入字节单元 DOWN 中。

答：
```
        JMP     BEGIN
UP      DB   0
DOWN    DB   0
TABLE   DB   100H DUP (?)     ；数组
BEGIN：
        MOV     CX，100H
        MOV     BX，-1
        MOV     SI，0
        MOV     DI，0
L1：    INC     BX
        CMP     TABLE[BX]，42H
        JA      L2
        JB      L3
        JMP     L4
L2：    INC     SI
        JMP     L4
L3：    INC     DI
L4：    LOOP    L1
        MOV     UP，SI
        MOV     DOWN，DI
```

3.52 请用图表示 ENTER 16，0 所生成的堆栈帧的情况。

答：答案见右图。



3.52 题的答案

第四章. 习 题

4.1 指出下列指令的错误：

```
(1)  MOV   AH, BX                          ；寄存器类型不匹配
(2)  MOV   [BX], [SI]                      ；不能都是存储器操作数
(3)  MOV   AX, [SI][DI]                    ；[SI]和[DI]不能一起使用
(4)  MOV   MYDAT  [BX][SI], ES:AX          ；AX 寄存器不能使用段超越
(5)  MOV   BYTE  PTR  [BX], 1000           ；1000 超过了一个字节的范围
(6)  MOV   BX, OFFSET  MYDAT  [SI]         ；MYDAT [SI]已经是偏移地址,不能再使用 OFFSET
(7)  MOV   CS, AX                          ；CS 不能用作目的寄存器
(8)  MOV   ECX, AX                         ；两个操作数的数据类型不同
```

答：见注释。

4.2 下面哪些指令是非法的？(假设 OP1，OP2 是已经用 DB 定义的变量)

```
(1)  CMP   15, BX                          ；错，立即数不能作为目的操作数
(2)  CMP   OP1, 25
```

(3) CMP OP1, OP2 ；错，不能都是存储器操作数
(4) CMP AX, OP1 ；错，类型不匹配，应为 CMP ax, word ptr op1
答：见注释。

4.3 假设下列指令中的所有标识符均为类型属性为字的变量，请指出下列哪些指令是非法的？它们的错误是什么？
(1) MOV BP, AL ；错，寄存器类型不匹配
(2) MOV WORD_OP [BX+4*3][DI], SP
(3) MOV WORD_OP1, WORD_OP2 ；错，不能都是存储器操作数
(4) MOV AX, WORD_OP1[DX] ；错，DX 不能用于存储器寻址
(5) MOV SAVE_WORD, DS
(6) MOV SP, SS:DATA_WORD [BX][SI]
(7) MOV [BX][SI], 2 ；错，[BX][SI]未指出数据类型
(8) MOV AX, WORD_OP1+WORD_OP2
(9) MOV AX, WORD_OP1-WORD_OP2+100
(10) MOV WORD_OP1, WORD_OP1-WORD_OP2
答：见注释。

4.4 假设 VAR1 和 VAR2 为字变量，LAB 为标号，试指出下列指令的错误之处：
(1) ADD VAR1, VAR2 ；不能都是存储器操作数
(2) SUB AL, VAR1 ；数据类型不匹配
(3) JMP LAB [SI] ；LAB 是标号而不是变量名，后面不能加[SI]
(4) JNZ VAR1 ；VAR1 是变量而不是标号
(5) JMP NEAR LAB ；应使用 NEAR PTR
答：见注释。

4.5 画图说明下列语句所分配的存储空间及初始化的数据值。
(1) BYTE_VAR DB 'BYTE',12,-12H,3 DUP(0,?,2 DUP(1,2),?)
(2) WORD_VAR DW 5 DUP(0,1,2),?,-5,'BY','TE',256H
答：答案如下图所示。

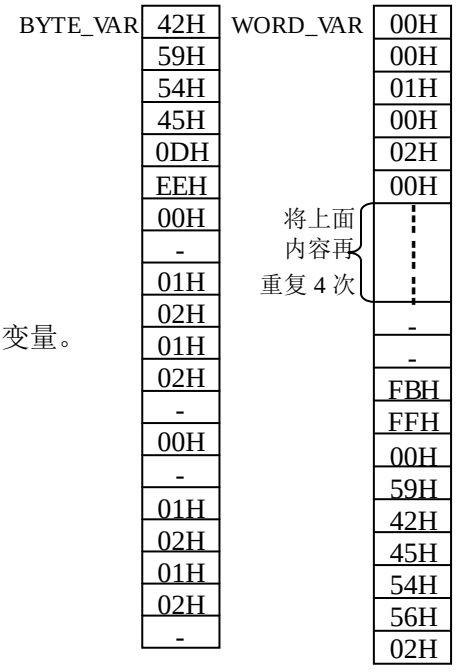
4.6 试列出各种方法，使汇编程序把 5150H 存入一个存储器字中(如：DW 5150H)。
答：DW 5150H

DB 50H, 51H

DB 'PQ'

DW 'QP'

ORG 5150H
DW $



4.5 题答案

4.7 请设置一个数据段 DATASG，其中定义以下字符变量或数据变量。
(1) FLD1B 为字符串变量：'personal computer'；
(2) FLD2B 为十进制数字节变量：32；
(3) FLD3B 为十六进制数字节变量：20；
(4) FLD4B 为二进制数字节变量：01011001；
(5) FLD5B 为数字的 ASCII 字符字节变量：32654；
(6) FLD6B 为 10 个零的字节变量；
(7) FLD7B 为零件名(ASCII 码)及其数量(十进制数)的表格：
PART1 20
PART2 50
PART3 14
(8) FLD1W 为十六进制数字变量：FFF0；
(9) FLD2W 为二进制数的字变量：01011001；
(10) FLD3W 为(7)零件表的地址变量；
(11) FLD4W 为包括 5 个十进制数的字变量：5，6，7，8，9；

(12) FLD5W 为 5 个零的字变量；
(13) FLD6W 为本段中字数据变量和字节数据变量之间的地址差。
答：
```
DATASG   SEGMENT
         FLD1B    DB   'personal computer'
         FLD2B    DB   32
         FLD3B    DB   20H
         FLD4B    DB   01011001B
         FLD5B    DB   '32654'
         FLD6B    DB   10  DUP (0)
         FLD7B    DB   'PART1', 20
                  DB   'PART2', 50
                  DB   'PART3', 14
         FLD1W    DW   0FFF0H
         FLD2W    DW   01011001B
         FLD3W    DW   FLD7B
         FLD4W    DW   5, 6, 7, 8, 9
         FLD5W    DW   5 DUP (0)
         FLD6W    DW   FLD1W-FLD1B
DATASG   ENDS
```

4.8 假设程序中的数据定义如下：
```
PARTNO  DW      ?
PNAME   DB      16 DUP (?)
COUNT   DD      ?
PLENTH  EQU     $-PARTNO
```
问 PLENTH 的值为多少？它表示什么意义？
答：PLENTH=22=16H，它表示变量 PARTNO、PNAME、COUNT 总共占用的存储单元数(字节数)。

4.9 有符号定义语句如下：
```
BUFF    DB    1, 2, 3, '123'
EBUFF   DB    0
L       EQU   EBUFF - BUFF
```
问 L 的值是多少？
答：L＝6。

4.10 假设程序中的数据定义如下：
```
LNAME       DB   30 DUP (?)
ADDRESS     DB   30 DUP (?)
CITY             DB   15 DUP (?)
CODE_LIST   DB   1, 7, 8, 3, 2
```
(1) 用一条 MOV 指令将 LNAME 的偏移地址放入 AX。
(2) 用一条指令将 CODE_LIST 的头两个字节的内容放入 SI。
(3) 用一条伪操作使 CODE_LENGTH 的值等于 CODE_LIST 域的实际长度。
答：
```
(1)  MOV   AX, OFFSET   LNAME
(2)  MOV   SI, WORD   PTR   CODE_LIST
(3)  CODE_LENGTH   EQU   $ - CODE_LIST   ；此语句必须放在 CODE_LIST 语句之后
```

4.11 试写出一个完整的数据段 DATA_SEG，它把整数 5 赋予一个字节，并把整数-1，0，2，5 和 4 放在 10 字数组 DATA_LIST 的头 5 个单元中。然后，写出完整的代码段，其功能为：把 DATA_LIST 中头 5 个数中的最大值和最小值分别存入 MAX 和 MIN 单元中。
答：
```
DATA_SEG    SEGMENT
  NUM       DB   5
  DATA_LIST DW   -1, 0, 2, 5, 4, 5 DUP (?)
  MAX       DW   ?
  MIN       DW   ?
DATA_SEG    ENDS
; ---------------------------------------------------------
CODE_SEG    SEGMENT
  MAIN      PROC   FAR
```

```
                ASSUME  CS: CODE_SEG, DS: DATA_SEG
START:          PUSH    DS                  ; 设置返回 DOS
                SUB     AX, AX
                PUSH    AX
                MOV     AX, DATA_SEG        ; 给 DS 赋值
                MOV     DS, AX
;
                MOV     CX, 4               ; 程序段开始
                LEA     BX, DATA_LIST
                MOV     AX, [BX]
                MOV     MAX, AX
                MOV     MIN, AX
ROUT1:          ADD     BX, 2
                MOV     AX, [BX]
                CMP     AX, MAX
                JNGE    ROUT2
                MOV     MAX, AX
ROUT2:          CMP     AX, MIN
                JNLE    ROUT3
                MOV     MIN, AX
ROUT3:          LOOP    ROUT1               ; 程序段结束
                RET
MAIN            ENDP
CODE_SEG        ENDS
; ---------------------------------------------------------
                END     START
```

4.12 给出等值语句如下：

```
ALPHA   EQU   100
BETA    EQU   25
GAMMA   EQU   2
```

下列表达式的值是多少？

(1) ALPHA * 100 + BETA ; =2729H
(2) ALPHA MOD GAMMA + BETA ; =19H
(3) (ALPHA +2) * BETA – 2 ; =9F4H
(4) (BETA / 3) MOD 5 ; =3H
(5) (ALPHA +3) * (BETA MOD GAMMA) ; =67H
(6) ALPHA GE GAMMA ; =0FFFFH
(7) BETA AND 7 ; =01H
(8) GAMMA OR 3 ; =03H

答：见注释。

4.13 对于下面的数据定义，三条 MOV 指令分别汇编成什么？(可用立即数方式表示)

```
TABLEA  DW    10 DUP (?)
TABLEB  DB    10 DUP (?)
TABLEC  DB    '1234'
  ⋮
MOV           AX, LENGTH TABLEA          ; 汇编成 MOV    AX, 000AH
MOV           BL, LENGTH TABLEB          ; 汇编成 MOV    BL, 000AH
MOV           CL, LENGTH TABLEC          ; 汇编成 MOV    CL, 0001H
```

答：见注释。

4.14 对于下面的数据定义，各条 MOV 指令单独执行后，有关寄存器的内容是什么？

```
FLDB    DB    ?
TABLEA  DW    20 DUP (?)
TABLEB  DB    'ABCD'
```

(1) MOV AX, TYPE FLDB ; (AX)=0001H
(2) MOV AX, TYPE TABLEA ; (AX)=0002H

```
(3)  MOV   CX, LENGTH  TABLEA        ; (CX)=0014H
(4)  MOV   DX, SIZE    TABLEA        ; (DX)=0028H
(5)  MOV   CX, LENGTH  TABLEB        ; (CX)=0001H
```

答：见注释。

4.15 指出下列伪操作表达方式的错误，并改正之。

```
(1)  DATA_SEG   SEG                  ; DATA_SEG SEGMENT（伪操作错）
(2)  SEGMENT    'CODE'               ; SEGNAME SEGMENT  'CODE'（缺少段名字）
(3)  MYDATA     SEGMENT/DATA         ; MYDATA   SEGMENT
                  ┇
                ENDS                 ; MYDATA   ENDS（缺少段名字）
(4)  MAIN_PROC PROC   FAR            ; 删除 END    MAIN_PROC 也可以
                  ┇
                END    MAIN_PROC     ; MAIN_PROC   ENDP   ；上下两句交换位置
     MAIN_PROC ENDP                  ;                  END    MAIN_PROC
```

答：见注释。

4.16 按下面的要求写出程序的框架

(1) 数据段的位置从 0E000H 开始，数据段中定义一个 100 字节的数组，其类型属性既是字又是字节；

(2) 堆栈段从小段开始，段组名为 STACK；

(3) 代码段中指定段寄存器，指定主程序从 1000H 开始，给有关段寄存器赋值；

(4) 程序结束。

答：程序的框架如下：

```
DATA_SEG    SEGMENT   AT   0E000H
  ARRAY_B   LABEL   BYTE
  ARRAY_W   DW      50 DUP (?)
DATA_SEG    ENDS                     ; 以上定义数据段
; ------------------------------------------------------
STACK_SEG   SEGMENT   PARA   STACK   'STACK'
            DW      100H DUP (?)
  TOS       LABEL   WORD
STACK_SEG   ENDS                     ; 以上定义堆栈段
; ------------------------------------------------------
CODE_SEG    SEGMENT
  MAIN      PROC    FAR
            ASSUME   CS: CODE_SEG,  DS: DATA_SEG,  SS: STACK_SEG
            ORG     1000H
  START:    MOV     AX, STACK_SEG
            MOV     SS, AX           ; 给 SS 赋值
            MOV     SP, OFFSET TOS   ; 给 SP 赋值
            PUSH    DS               ; 设置返回 DOS
            SUB          AX, AX
            PUSH    AX
            MOV     AX, DATA_SEG
            MOV     DS, AX           ; 给 DS 赋值
              ┇                      ; 程序段部分
            RET
  MAIN      ENDP
CODE_SEG    ENDS                     ; 以上定义代码段
; ------------------------------------------------------
            END     START
```

4.17 写一个完整的程序放在代码段 C_SEG 中，要求把数据段 D_SEG 中的 AUGEND 和附加段 E_SEG 中的 ADDEND 相加，并把结果存放在 D_SEG 段中的 SUM 中。其中 AUGEND、ADDEND 和 SUM 均为双精度数，AUGEND 赋值为 99251，ADDEND 赋值为 -15962。

答：程序如下：

```
D_SEG     SEGMENT
  AUGW    LABEL   WORD
  AUGEND  DD      99251
  SUM     DD      ?
D_SEG     ENDS                    ；以上定义数据段
；-------------------------------------------------------
E_SEG     SEGMENT
  ADDW    LABEL   WORD
  ADDEND  DD      -15962
E_SEG     ENDS                    ；以上定义附加段
；-------------------------------------------------------
C_SEG     SEGMENT
  MAIN    PROC    FAR
          ASSUME  CS: C_SEG, DS: D_SEG, ES: E_SEG
  START:  PUSH    DS              ；设置返回 DOS
          SUB       AX, AX
          PUSH    AX
          MOV     AX, D_SEG
          MOV     DS, AX          ；给 DS 赋值
          MOV     AX, E_SEG
          MOV     ES, AX          ；给 ES 赋值
  ；
          MOV     AX, AUGW        ；以下 6 条指令进行加法计算
          MOV     BX, AUGW+2
          ADD     AX, ES: ADDW
          ADC     BX, ES: ADDW+2  ；不考虑有符号数溢出
          MOV     WORD PTR SUM, AX
          MOV     WORD PTR [SUM+2], BX
          RET
  MAIN    ENDP
C_SEG     ENDS                    ；以上定义代码段
；-------------------------------------------------------
          END     START
```

4.18 请说明表示程序结束的微操作和结束程序执行的语句之间的差别。它们在源程序中应如何表示？

答：表示程序结束的微操作是指示汇编程序 MASM 结束汇编的标志，在源程序中用 END 表示；结束程序执行的语句是结束程序运行而返回操作系统的指令，在源程序中有多种表示方法，比如 INT 20H 或 MOV AX, 4C00H　　INT　21H 以及 RET 等。

4.19 试说明下述指令中哪些需要加上 PTR 操作符：

```
BVAL    DB   10H，20H
WVAL    DW   1000H
(1)  MOV    AL，BVAL                ；不需要
(2)  MOV    DL，[BX]                ；不需要
(3)  SUB    [BX]，2                 ；需要，如 SUB　BYTE PTR [BX]，2
(4)  MOV    CL，WVAL                ；需要，如 MOV CL，BYTE PTR WVAL
(5)  ADD    AL，BVAL+1              ；不需要
```

答：见注释。

第五章. 习　题

5.1 试编写一个汇编语言程序，要求对键盘输入的小写字母用大写字母显示出来。

答：程序段如下：

```
BEGIN:  MOV     AH, 1           ；从键盘输入一个字符的 DOS 调用
        INT     21H
        CMP     AL, ‘a’         ；输入字符<‘a’吗？
```

```
        JB      STOP
        CMP     AL, 'z'         ; 输入字符>'z'吗?
        JA      STOP
        SUB     AL, 20H         ; 转换为大写字母，用 AND AL, 1101 1111B 也可
        MOV     DL, AL          ; 显示一个字符的 DOS 调用
        MOV     AH, 2
        INT     21H
        JMP     BEGIN
STOP:   RET
```

5.2 编写程序，从键盘接收一个小写字母，然后找出它的前导字符和后续字符，再按顺序显示这三个字符。

答：程序段如下：

```
BEGIN:  MOV     AH, 1           ; 从键盘输入一个字符的 DOS 调用
        INT     21H
        CMP     AL, 'a'         ; 输入字符<'a'吗?
        JB      STOP
        CMP     AL, 'z'         ; 输入字符>'z'吗?
        JA      STOP
        DEC     AL              ; 得到前导字符
        MOV     DL, AL          ; 准备显示三个字符
        MOV     CX, 3
DISPLAY: MOV    AH, 2           ; 显示一个字符的 DOS 调用
        INT     21H
        INC     DL
        LOOP    DISPLAY
STOP:   RET
```

5.3 将 AX 寄存器中的 16 位数分成 4 组，每组 4 位，然后把这四组数分别放在 AL、BL、CL 和 DL 中。

答：程序段如下：

```
DSEG    SEGMENT
  STORE DB  4 DUP (?)
DSEG    ENDS
        ┇
BEGIN:  MOV     CL, 4           ; 右移四次
        MOV     CH, 4           ; 循环四次
        LEA     BX, STORE
A10:    MOV     DX, AX
        AND     DX, 0FH         ; 取 AX 的低四位
        MOV     [BX], DL        ; 低四位存入 STORE 中
        INC     BX
        SHR     AX, CL          ; 右移四次
        DEC     CH
        JNZ     A10             ; 循环四次完了码?
B10:    MOV     DL, STORE       ; 四组数分别放在 AL、BL、CL 和 DL 中
        MOV     CL, STORE+1
        MOV     BL, STORE+2
        MOV     AL, STORE+3
STOP:   RET
```

5.4 试编写一程序，要求比较两个字符串 STRING1 和 STRING2 所含字符是否完全相同，若相同则显示'MATCH'， 若不相同则显示'NO MATCH'。

答：程序如下：

```
DSEG    SEGMENT
  STRING1 DB  'I am a student.'
  STRING2 DB  'I am a student!'
  YES     DB  'MATCH', 0DH, 0AH, '$'
```

```
  NO          DB   'NO MATCH', 0DH, 0AH, '$'
DSEG          ENDS
; ------------------------------------------------------------
CSEG          SEGMENT
  MAIN        PROC   FAR
              ASSUME  CS: CSEG, DS: DSEG, ES: DSEG
  START:      PUSH   DS                ; 设置返回 DOS
              SUB        AX, AX
              PUSH   AX
              MOV    AX, DSEG
              MOV    DS, AX            ; 给 DS 赋值
              MOV    ES, AX            ; 给 ES 赋值
  ;
  BEGIN:      LEA    SI, STRING1       ; 设置串比较指令的初值
              LEA        DI, STRING2
              CLD
              MOV    CX, STRING2 - STRING1
              REPE   CMPSB             ; 串比较
              JNE    DISPNO
              LEA        DX, YES           ; 显示 MATCH
              JMP    DISPLAY
  DISPNO:     LEA    DX, NO            ; 显示 NO MATCH
  DISPLAY:    MOV    AH, 9             ; 显示一个字符串的 DOS 调用
              INT    21H
              RET
  MAIN        ENDP
CSEG          ENDS                     ; 以上定义代码段
; ------------------------------------------------------------
              END    START
```

5.5 试编写一程序，要求能从键盘接收一个个位数 N，然后响铃 N 次(响铃的 ASCII 码为 07)。

答：程序段如下：

```
  BEGIN:      MOV    AH, 1             ; 从键盘输入一个字符的 DOS 调用
              INT    21H
              SUB        AL, '0'
              JB     STOP              ; 输入字符<'0'吗？
              CMP    AL, 9             ; 输入字符>'9'吗？
              JA     STOP
              CBW
              MOV    CX, AX            ; 响铃次数 N
              JCXZ   STOP
  BELL:       MOV    DL, 07H           ; 准备响铃
              MOV    AH, 2             ; 显示一个字符的 DOS 调用，实际为响铃
              INT    21H
              CALL   DELAY100ms        ; 延时 100ms
              LOOP   BELL
  STOP:       RET
```

5.6 编写程序，将一个包含有 20 个数据的数组 M 分成两个数组：正数数组 P 和负数数组 N，并分别把这两个数组中数据的个数显示出来。

答：程序如下：

```
DSEG          SEGMENT
  COUNT           EQU   20
  ARRAY           DW    20 DUP (?)          ; 存放数组
  COUNT1      DB    0                 ; 存放正数的个数
  ARRAY1      DW    20 DUP (?)        ; 存放正数
  COUNT2      DB    0                 ; 存放负数的个数
```

```
  ARRAY2    DW   20 DUP (?)            ; 存放负数
  ZHEN      DB   0DH, 0AH, 'The positive number is：', '$'     ; 正数的个数是：
  FU        DB   0DH, 0AH, 'The negative number is：', '$'     ; 负数的个数是：
  CRLF      DB   0DH, 0AH, '$'
DSEG        ENDS
; ---------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC   FAR
            ASSUME  CS: CSEG, DS: DSEG
  START:    PUSH   DS              ; 设置返回 DOS
            SUB        AX, AX
            PUSH   AX
            MOV    AX, DSEG
            MOV    DS, AX          ; 给 DS 赋值

  BEGIN:    MOV    CX, COUNT
            LEA        BX, ARRAY
            LEA        SI, ARRAY1
            LEA        DI, ARRAY2
  BEGIN1:   MOV    AX, [BX]
            CMP    AX, 0           ; 是负数码?
            JS     FUSHU
            MOV    [SI], AX        ; 是正数，存入正数数组
            INC    COUNT1          ; 正数个数+1
            ADD    SI, 2
            JMP    SHORT  NEXT
  FUSHU:    MOV    [DI], AX        ; 是负数，存入负数数组
            INC    COUNT2          ; 负数个数+1
            ADD    DI, 2
  NEXT:     ADD    BX, 2
            LOOP   BEGIN1
            LEA        DX, ZHEN        ; 显示正数个数
            MOV    AL, COUNT1
            CALL   DISPLAY         ; 调显示子程序
            LEA        DX, FU          ; 显示负数个数
            MOV    AL, COUNT2
            CALL   DISPLAY         ; 调显示子程序
            RET
  MAIN      ENDP
; ---------------------------------------------------------------
  DISPLAY   PROC   NEAR            ; 显示子程序
            MOV    AH, 9           ; 显示一个字符串的 DOS 调用
            INT    21H
            AAM                    ; 将(AL)中的二进制数转换为二个非压缩 BCD 码
            ADD    AH, '0'         ; 变为 0～9 的 ASCII 码
            MOV    DL, AH
            MOV    AH, 2           ; 显示一个字符的 DOS 调用
            INT    21H
            ADD    AL, '0'         ; 变为 0～9 的 ASCII 码
            MOV    DL, AL
            MOV    AH, 2           ; 显示一个字符的 DOS 调用
            INT    21H
            LEA        DX, CRLF        ; 显示回车换行
            MOV    AH, 9           ; 显示一个字符串的 DOS 调用
            INT    21H
            RET
  DISPLAY   ENDP                   ; 显示子程序结束
```

```
CSEG        ENDS                        ；以上定义代码段
；-------------------------------------------------------------
            END     START
```

5.7 试编写一个汇编语言程序，求出首地址为 DATA 的 100D 字数组中的最小偶数，并把它存放在 AX 中。

答：程序段如下：

```
BEGIN:      MOV     BX, 0
            MOV     CX, 100
COMPARE:    MOV     AX, DATA[BX]    ；取数组的第一个偶数
            ADD     BX, 2
            TEST    AX, 01H         ；是偶数吗？
            LOOPNZ COMPARE          ；不是，比较下一个数
            JNZ     STOP            ；没有偶数，退出
            JCXZ    STOP            ；最后一个数是偶数，即为最小偶数，退出
COMPARE1:       MOV     DX, DATA[BX]    ；取数组的下一个偶数
            ADD     BX, 2
            TEST    DX, 01H         ；是偶数吗？
            JNZ     NEXT            ；不是，比较下一个数
            CMP     AX, DX          ；(AX)<(DX)吗？
            JLE     NEXT
            MOV     AX, DX          ；(AX)<(DX)，则置换(AX)为最小偶数
NEXT:       LOOP    COMPARE1
STOP:       RET
```

5.8 把 AX 中存放的 16 位二进制数 K 看作是 8 个二进制的“四分之一字节”。试编写程序要求数一下值为 3(即 11B)的四分之一字节数，并将该数(即 11B 的个数)在终端上显示出来。

答：程序段如下：

```
BEGIN:      MOV     DL, 0           ；计数初始值
            MOV     CX, 8
COMPARE:    TEST    AX, 03H         ；是数 03 吗？
            JNZ     NOEQUAL         ；不是，转走
            INC     DL              ；是，计数
NOEQUAL:    ROR     AX, 1           ；准备判断下一个数
            ROR     AX, 1
            LOOP    COMPARE
            ADD     DL, ‘0’         ；将计数值转换为 ASCII 码
            MOV     AH, 2           ；进行显示
            INT     21H
STOP:       RET
```

5.9 试编写一个汇编语言程序，要求从键盘接收一个四位的 16 进制数，并在终端上显示与它等值的二进制数。

答：程序段如下：

```
BEGIN:      MOV     BX, 0           ；用于存放四位的 16 进制数
            MOV     CH, 4
            MOV     CL, 4
INPUT:      SHL     BX, CL          ；将前面输入的数左移 4 位
            MOV     AH, 1           ；从键盘取数
            INT     21H
            CMP     AL, 30H         ；<0 吗？
            JB      INPUT           ；不是‘0～F’的数重新输入
            CMP     AL, 39H         ；是‘0～9’吗？
            JA      AF              ；不是，转‘A～F’的处理
            AND     AL, 0FH         ；转换为：0000B～1001B
            JMP     BINARY
AF:         AND     AL, 1101 1111B  ；转换为大写字母
```

```
                CMP     AL, 41H         ；又<A 吗？
                JB      INPUT           ；不是‘A～F’的数重新输入
                CMP     AL, 46H         ；>F 吗？
                JA      INPUT           ；不是‘A～F’的数重新输入
                AND     AL, 0FH         ；转换为：1010B～1111B
                ADD     AL, 9
BINARY:         OR      BL, AL          ；将键盘输入的数进行组合
                DEL     CH
                JNZ     INPUT
DISPN:          MOV     CX, 16          ；将 16 位二进制数一位位地转换成 ASCII 码显示
DISP:           MOV     DL, 0
                ROL     BX, 1
                RCL     DL, 1
                OR      DL, 30H
                MOV     AH, 2           ；进行显示
                INT     21H
                LOOP    DISP
STOP:           RET
```

5.10 设有一段英文，其字符变量名为 ENG，并以$字符结束。试编写一程序，查对单词 SUN 在该文中的出现次数，并以格式“SUN：xxxx”显示出次数。

答：程序如下：

```
DSEG        SEGMENT
  ENG       DB  ‘Here is sun, sun ,…,$’
  DISP      DB  ‘SUN：’
  DAT       DB  ‘0000’, 0DH, 0AH, ‘$’
  KEYWORD   DB  ‘sun’
DSEG        ENDS
；-------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME  CS: CSEG, DS: DSEG, ES: DSEG
  START:    PUSH    DS              ；设置返回 DOS
            SUB     AX, AX
            PUSH    AX
            MOV     AX, DSEG
            MOV     DS, AX          ；给 DS 赋值
            MOV     ES, AX          ；给 ES 赋值

  BEGIN:    MOV     AX, 0
            MOV     DX, DISP-ENG-2  ；计算 ENG 的长度(每次比较 sun,因此比较次数-2)
            LEA     BX, ENG
  COMP:     MOV     DI, BX
            LEA     SI, KEYWORD
            MOV     CX, 3
            REPE    CMPSB           ；串比较
            JNZ     NOMATCH
            INC     AX              ；是，SUN 的个数加 1
            ADD     BX, 2
  NOMATCH:  INC     BX              ；指向 ENG 的下一个字母
            DEC     DX
            JNZ     COMP
  DONE:     MOV     CH, 4           ；将次数转换为 16 进制数的 ASCII 码
            MOV     CL, 4
            LEA     BX, DAT             ；转换结果存入 DAT 单元中
  DONE1:    ROL     AX, CL
            MOV     DX, AX
            AND     DL, 0FH         ；取一位 16 进制数
```

```
                ADD     DL, 30H
                CMP     DL, 39H
                JLE     STORE
                ADD     DL, 07H          ; 是“A～F”所以要加 7
  STORE:        MOV     [BX], DL         ; 转换结果存入 DAT 单元中
                INC     BX
                DEC     CH
                JNZ     DONE1
  DISPLAY:      LEA     DX, DISP         ; 显示字符串程序(将 DISP 和 DAT 一起显示)
                MOV     AH, 09H
                INT     21H
                RET
  MAIN          ENDP
CSEG            ENDS                     ; 以上定义代码段
; ------------------------------------------------------------
                END     START
```

5.11 从键盘输入一系列以$为结束符的字符串，然后对其中的非数字字符计数，并显示出计数结果。

答：程序段如下：

```
DSEG            SEGMENT
  BUFF          DB  50 DUP (' ')
  COUNT         DW   0
DSEG            ENDS
                 ┇
  BEGIN:        LEA     BX, BUFF
                MOV     COUNT, 0
  INPUT:        MOV     AH, 01           ; 从键盘输入一个字符的功能调用
                INT     21H
                MOV     [BX], AL
                INC     BX
                CMP     AL, '$'          ; 是$结束符吗？
                JNZ     INPUT            ; 不是，继续输入
                LEA     BX, BUFF         ; 对非数字字符进行计数
  NEXT:         MOV     CL, [BX]
                INC     BX
                CMP     CL, '$'          ; 是$结束符，则转去显示
                JZ      DISP
                CMP     CL, 30H          ; 小于 0 是非数字字符
                JB      NEXT
                CMP     CL, 39H          ; 大于 9 是非数字字符
                JA      NEXT
                INC     COUNT            ; 个数+1
                JMP     NEXT
  DISP:          ┇                       ; 16 进制数显示程序段(省略)
```

5.12 有一个首地址为 MEM 的 100D 字数组，试编制程序删除数组中所有为 0 的项，并将后续项向前压缩，最后将数组的剩余部分补上 0。

答：程序如下：

```
DSEG            SEGMENT
  MEM           DW   100 DUP (?)
DSEG            ENDS
; ------------------------------------------------------------
CSEG            SEGMENT
  MAIN          PROC    FAR
                ASSUME  CS: CSEG, DS: DSEG
  START:        PUSH    DS               ; 设置返回 DOS
                SUB     AX, AX
                PUSH    AX
```

```
                MOV     AX, DSEG
                MOV     DS, AX          ; 给 DS 赋值

  BEGIN:        MOV     SI, (100-1)*2   ; (SI)指向 MEM 的末元素的首地址
                MOV     BX, -2          ; 地址指针的初值
                MOV     CX, 100
  COMP:         ADD     BX, 2
                CMP     MEM [BX], 0
                JZ      CONS
                LOOP    COMP
                JMP     FINISH          ; 比较完了，已无 0 则结束
  CONS:         MOV     DI, BX
  CONS1:        CMP     DI, SI          ; 到了最后单元码?
                JAE     NOMOV
                MOV     AX, MEM [DI+2]  ; 后面的元素向前移位
                MOV     MEM [DI], AX
                ADD     DI, 2
                JMP     CONS1
  NOMOV:        MOV     WORD PTR [SI], 0; 最后单元补 0
                LOOP    COMP
  FINISH:       RET
  MAIN          ENDP
CSEG            ENDS                    ; 以上定义代码段
; ---------------------------------------------------------------
                END     START
```

5.13 在 STRING 到 STRING+99 单元中存放着一个字符串，试编制一个程序测试该字符串中是否存在数字，如有则把 CL 的第 5 位置 1，否则将该位置 0。

答：程序如下：

```
DSEG            SEGMENT
  STRING        DB    100 DUP (?)
DSEG            ENDS
; ---------------------------------------------------------------
CSEG            SEGMENT
  MAIN          PROC    FAR
                ASSUME    CS: CSEG, DS: DSEG
  START:        PUSH    DS              ; 设置返回 DOS
                SUB       AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX          ; 给 DS 赋值

  BEGIN:        MOV     SI, 0           ; (SI)作为地址指针的变化值
                MOV     CX, 100
  REPEAT:       MOV     AL, STRING [SI]
                CMP     AL, 30H
                JB      GO_ON
                CMP     AL, 39H
                JA      GO_ON
                OR      CL, 20H         ; 存在数字把 CL 的第 5 位置 1
                JMP     EXIT
  GO_ON:        INC     SI
                LOOP    REPEAT
                AND     CL, 0DFH        ; 不存在数字把 CL 的第 5 位置 0
  EXIT:         RET
  MAIN          ENDP
CSEG            ENDS                    ; 以上定义代码段
; ---------------------------------------------------------------
```

```
                END     START
```

5.14 在首地址为 TABLE 的数组中按递增次序存放着 100H 个 16 位补码数，试编写一个程序把出现次数最多的数及其出现次数分别存放于 AX 和 CX 中。

答：程序如下：

```
DSEG        SEGMENT
  TABLE     DW   100H DUP (?)        ；数组中的数据是按增序排列的
  DATA      DW   ?
  COUNT         DW   0
DSEG        ENDS
；-------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME   CS: CSEG, DS: DSEG
  START:    PUSH    DS              ；设置返回 DOS
            SUB          AX, AX
            PUSH    AX
            MOV     AX, DSEG
            MOV     DS, AX          ；给 DS 赋值

  BEGIN:    MOV     CX, 100H        ；循环计数器
            MOV     SI, 0
  NEXT:     MOV     DX, 0
            MOV     AX, TABLE [SI]
  COMP:     CMP     TABLE [SI], AX  ；计算一个数的出现次数
            JNE     ADDR
            INC     DX
                ADD     SI, 2
            LOOP    COMP
  ADDR:     CMP     DX, COUNT       ；此数出现的次数最多吗？
            JLE     DONE
            MOV     COUNT, DX       ；目前此数出现的次数最多，记下次数
            MOV     DATA, AX        ；记下此数
  DONE:     LOOP    NEXT            ；准备取下一个数
            MOV     CX, COUNT       ；出现最多的次数存入(CX)
            MOV     AX, DATA        ；出现最多的数存入(AX)
            RET
  MAIN      ENDP
CSEG        ENDS                    ；以上定义代码段
；-------------------------------------------------------------
            END     START
```

5.15 数据段中已定义了一个有 n 个字数据的数组 M，试编写一程序求出 M 中绝对值最大的数，把它放在数据段的 M+2n 单元中，并将该数的偏移地址存放在 M+2(n+1)单元中。

答：程序如下：

```
DSEG        SEGMENT
  n         EQU   100H              ；假设 n=100H
  M         DW    n DUP (?)
  DATA      DW    ?                 ；M+2n 单元
  ADDR      DW    ?                 ；M+2(n+1)单元
DSEG        ENDS
；-------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME   CS: CSEG, DS: DSEG
  START:    PUSH    DS              ；设置返回 DOS
            SUB          AX, AX
            PUSH    AX
```

```
                MOV     AX, DSEG
                MOV     DS, AX          ; 给 DS 赋值

BEGIN:          MOV     CX, n           ; 循环计数器
                LEA     DI, M
                MOV     AX, [DI]        ; 取第一个数
                MOV     ADDR, DI        ; 记下绝对值最大的数的地址
                CMP     AX, 0           ; 此数是正数吗?
                JNS     ZHEN            ; 是正数，即为绝对值，转去判断下一个数
                NEG     AX              ; 不是正数，变为其绝对值
ZHEN:           MOV     BX, [DI]
                CMP     BX, 0           ; 此数是正数吗?
                JNS     COMP            ; 是正数，即为绝对值，转去比较绝对值大小
                NEG     BX              ; 不是正数，变为其绝对值
COMP:           CMP     AX, BX          ; 判断绝对值大小
                JAE     ADDRESS
                MOV     AX, BX          ; (AX)<(BX)，使(AX)中为绝对值最大的数
                MOV     ADDR, DI        ; 记下绝对值最大的数的地址
ADDRESS:        ADD     DI, 2
                LOOP    ZHEN
                MOV     DATA, AX        ; 记下此数
                RET
MAIN            ENDP
CSEG            ENDS                    ; 以上定义代码段
; ------------------------------------------------------------
                END     START
```

5.16 在首地址为 DATA 的字数组中存放着 100H 个 16 位补码数，试编写一个程序求出它们的平均值放在 AX 寄存器中；并求出数组中有多少个数小于此平均值，将结果放在 BX 寄存器中。

答：程序如下：

```
DSEG            SEGMENT
  DATA          DW   100H DUP (?)
DSEG            ENDS
; ------------------------------------------------------------
CSEG            SEGMENT
  MAIN          PROC    FAR
                ASSUME  CS: CSEG, DS: DSEG
  START:        PUSH    DS              ; 设置返回 DOS
                SUB     AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX          ; 给 DS 赋值

  BEGIN:        MOV     CX, 100H        ; 循环计数器
                MOV     SI, 0
                MOV     BX, 0           ; 和((DI),(BX))的初始值
                MOV     DI, 0
  NEXT:         MOV     AX, DATA [SI]
                CWD
                ADD     BX, AX          ; 求和
                ADC     DI, DX          ; 加上进位位
                ADD     SI, 2
                LOOP    NEXT
                MOV     DX, DI          ; 将((DI),(BX))中的累加和放入((DX),(AX))中
                MOV     AX, BX
                MOV     CX, 100H
                IDIV    CX              ; 带符号数求平均值，放入(AX)中
```

```
                MOV     BX, 0
                MOV     SI, 0
  COMP:         CMP     AX, DATA [SI]     ；寻找小于平均值的数
                JLE     NO
                INC     BX                ；小于平均值数的个数+1
  NO:           ADD     SI, 2
                LOOP    COMP
                RET
  MAIN          ENDP
CSEG            ENDS                      ；以上定义代码段
；--------------------------------------------------------------------
                END     START
```

5.17 试编制一个程序把 AX 中的 16 进制数转换为 ASCII 码，并将对应的 ASCII 码依次存放到 MEM 数组中的四个字节中。例如，当(AX)=2A49H 时，程序执行完后，MEM 中的 4 个字节内容为 39H，34H，41H，32H。

答：程序如下：

```
DSEG            SEGMENT
  MEM           DB    4 DUP (?)
  N             DW    2A49H
DSEG            ENDS
；--------------------------------------------------------------------
CSEG            SEGMENT
  MAIN          PROC    FAR
                ASSUME    CS: CSEG, DS: DSEG
  START:        PUSH    DS                ；设置返回 DOS
                SUB           AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX            ；给 DS 赋值

  BEGIN:        MOV     CH, 4             ；循环计数器
                MOV     CL, 4
                MOV     AX, N
                LEA           BX, MEM
  ROTATE:       MOV     DL, AL            ；从最低四位开始转换为 ASCII 码
                AND     DL, 0FH
                ADD     DL, 30H
                CMP     DL, 3AH           ；是 0~9 吗？
                JL      NEXT
                ADD     DL, 07H           ；是 A~F
  NEXT:         MOV     [BX], DL          ；转换的 ASCII 码送入 MEM 中
                INC     BX
                ROR     AX, CL            ；准备转换下一位
                DEC           CH
                JNZ     ROTATE
                RET
  MAIN          ENDP
CSEG            ENDS                      ；以上定义代码段
；--------------------------------------------------------------------
                END     START
```

5.18 把 0~100D 之间的 30 个数存入以 GRADE 为首地址的 30 字数组中，GRADE+i 表示学号为 i+1 的学生的成绩。另一个数组 RANK 为 30 个学生的名次表，其中 RANK+i 的内容是学号为 i+1 的学生的名次。编写一程序，根据 GRADE 中的学生成绩，将学生名次填入 RANK 数组中。(提示：一个学生的名次等于成绩高于这个学生的人数加 1。)

答：程序如下：

```
DSEG            SEGMENT
```

```
  GRADE       DW   30 DUP (?)          ；假设已预先存好 30 名学生的成绩
  RANK        DW   30 DUP (?)
DSEG          ENDS
；-------------------------------------------------------------
CSEG          SEGMENT
  MAIN        PROC   FAR
              ASSUME  CS: CSEG, DS: DSEG
  START:      PUSH   DS              ；设置返回 DOS
              SUB        AX, AX
              PUSH   AX
              MOV    AX, DSEG
              MOV    DS, AX          ；给 DS 赋值

  BEGIN:      MOV    DI, 0
              MOV    CX, 30          ；外循环计数器
  LOOP1:      PUSH   CX
              MOV    CX, 30          ；内循环计数器
              MOV    SI, 0
              MOV    AX, GRADE [DI]
              MOV    DX, 1           ；起始名次为第 1 名
  LOOP2:      CMP    GRADE [SI], AX  ；成绩比较
              JBE    GO_ON
              INC    DX              ；名次+1
  GO_ON:      ADD    SI, 2
              LOOP   LOOP2
              POP    CX
              MOV    RNAK [DI], DX   ；名次存入 RANK 数组
              ADD    DI, 2
              LOOP   LOOP1
              RET
  MAIN        ENDP
CSEG          ENDS                   ；以上定义代码段
；-------------------------------------------------------------
              END    START
```

5.19 已知数组 A 包含 15 个互不相等的整数，数组 B 包含 20 个互不相等的整数。试编制一程序把既在 A 中又在 B 中出现的整数存放于数组 C 中。

答：程序如下：

```
DSEG          SEGMENT
  A           DW   15 DUP (?)
  B           DW   20 DUP (?)
  C           DW   15 DUP (‘ ’)
DSEG          ENDS
；-------------------------------------------------------------
CSEG          SEGMENT
  MAIN        PROC   FAR
              ASSUME  CS: CSEG, DS: DSEG
  START:      PUSH   DS              ；设置返回 DOS
              SUB        AX, AX
              PUSH   AX
              MOV    AX, DSEG
              MOV    DS, AX          ；给 DS 赋值

  BEGIN:      MOV    SI, 0
              MOV    BX, 0
              MOV    CX, 15          ；外循环计数器
  LOOP1:      PUSH   CX
              MOV    CX, 20          ；内循环计数器
```

```
                MOV     DI, 0
                MOV     AX, A [SI]          ；取 A 数组中的一个数
  LOOP2:        CMP     B [DI], AX          ；和 B 数组中的数相等吗？
                JNE     NO
                MOV     C [BX], AX          ；相等存入 C 数组中
                ADD     BX, 2
  NO:           ADD     DI, 2
                LOOP    LOOP2
                ADD     SI, 2
                POP     CX
                LOOP    LOOP1
                RET
  MAIN          ENDP
CSEG            ENDS                        ；以上定义代码段
；-------------------------------------------------------------
                END     START
```

5.20 设在 A、B 和 C 单元中分别存放着三个数。若三个数都不是 0，则求出三数之和存放在 D 单元中；若其中有一个数为 0，则把其它两单元也清 0。请编写此程序。

答：程序如下：

```
DSEG            SEGMENT
  A             DW   ?
  B             DW   ?
  C             DW   ?
  D             DW   0
DSEG            ENDS
；-------------------------------------------------------------
CSEG            SEGMENT
  MAIN          PROC    FAR
                ASSUME  CS: CSEG, DS: DSEG
  START:        PUSH    DS                  ；设置返回 DOS
                SUB         AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX              ；给 DS 赋值

  BEGIN:        CMP     A, 0
                JE      NEXT
                CMP     B, 0
                JE      NEXT
                CMP     C, 0
                JE      NEXT
                MOV     AX, A
                ADD     AX, B
                ADD     AX, C
                MOV     D, AX
                JMP     SHORT   EXIT
  NEXT:         MOV     A, 0
                MOV     B, 0
                MOV     C, 0
  EXIT:         RET
  MAIN          ENDP
CSEG            ENDS                        ；以上定义代码段
；-------------------------------------------------------------
                END     START
```

5.21 试编写一程序，要求比较数组 ARRAY 中的三个 16 位补码数，并根据比较结果在终端上显示如下信息：

(1) 如果三个数都不相等则显示 0；

(2) 如果三个数有二个数相等则显示 1；
(3) 如果三个数都相等则显示 2。
答：程序如下：

```
DSEG        SEGMENT
  ARRAY         DW   3 DUP (?)
DSEG        ENDS
;  ------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME   CS: CSEG, DS: DSEG
  START:    PUSH    DS              ；设置返回 DOS
            SUB         AX, AX
            PUSH    AX
            MOV     AX, DSEG
            MOV     DS, AX          ；给 DS 赋值

  BEGIN:    LEA     SI, ARRAY
            MOV     DX, 0           ；(DX)用于存放所求的结果
            MOV     AX, [SI]
            MOV     BX, [SI+2]
            CMP     AX, BX          ；比较第一和第二两个数是否相等
            JNE     NEXT1
            INC     DX
  NEXT1:        CMP     [SI+4], AX      ；比较第一和第三两个数是否相等
            JNE     NEXT2
            INC     DX
  NEXT2:        CMP     [SI+4], BX      ；比较第二和第三两个数是否相等
            JNE     NUM
            INC     DX
  NUM:      CMP     DX, 3
            JL      DISP
            DEC         DX
  DISP:     ADD     DL, 30H         ；转换为 ASCII 码
            MOV     AH, 2           ；显示一个字符
            INT     21H
            RET
  MAIN      ENDP
CSEG        ENDS                    ；以上定义代码段
;  ------------------------------------------------------------
            END     START
```

5.22 从键盘输入一系列字符(以回车符结束)，并按字母、数字、及其它字符分类计数，最后显示出这三类的计数结果。
答：程序如下：

```
DSEG        SEGMENT
  ALPHABET  DB   ‘输入的字母字符个数为：’, ‘$’
  NUMBER    DB   ‘输入的数字字符个数为：’, ‘$’
  OTHER     DB   ‘输入的其它字符个数为：’, ‘$’
  CRLF      DB   0DH, 0AH, ‘$’
DSEG        ENDS
;  ------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME   CS: CSEG, DS: DSEG
  START:    PUSH    DS              ；设置返回 DOS
            SUB         AX, AX
            PUSH    AX
            MOV     AX, DSEG
```

```
                MOV     DS, AX              ; 给 DS 赋值

BEGIN:          MOV     BX, 0               ; 字母字符计数器
                MOV     SI, 0               ; 数字字符计数器
                MOV     DI, 0               ; 其它字符计数器
INPUT:          MOV     AH, 1               ; 输入一个字符
                INT     21H
                CMP     AL, 0DH             ; 是回车符吗?
                JE      DISP
                CMP     AL, 30H             ; <数字 0 吗?
                JAE     NEXT1
OTHER:          INC     DI                  ; 是其它字符
                JMP     SHORT   INPUT
NEXT1:          CMP     AL, 39H             ; >数字 9 吗?
                JA      NEXT2
                INC     SI                  ; 是数字字符
                JMP     SHORT   INPUT
NEXT2:          CMP     AL, 41H             ; <字母 A 吗?
                JAE     NEXT3
                JMP     SHORT OTHER         ; 是其它字符
NEXT3:          CMP     AL, 5AH             ; >字母 Z 吗?
                JA      NEXT4
                INC     BX                  ; 是字母字符 A~Z
                JMP     SHORT   INPUT
NEXT4:          CMP     AL, 61H             ; <字母 a 吗?
                JAE     NEXT5
                JMP     SHORT OTHER         ; 是其它字符
NEXT5:          CMP     AL, 7AH             ; >字母 z 吗?
                JA      SHORT OTHER         ; 是其它字符
                INC     BX                  ; 是字母字符 a~z
                JMP     SHORT   INPUT

DISP:           LEA     DX, ALPHABET
                CALL    DISPLAY
                LEA     DX, NUMBER
                MOV     BX, SI
                CALL    DISPLAY
                LEA     DX, OTHER
                MOV     BX, DI
                CALL    DISPLAY
                RET
MAIN            ENDP
; ---------------------------------------------------------------------
DISPLAY         PROC    NEAR
                MOV     AH, 09H             ; 显示字符串功能调用
                INT     21H
                CALL    BINIHEX             ; 调把 BX 中二进制数转换为 16 进制显示子程序
                LEA     DX, CRLF
                MOV     AH, 09H             ; 显示回车换行
                INT     21H
                RET
DISPLAY         ENDP
; ---------------------------------------------------------------------
BINIHEX         PROC    NEAR                ; 将 BX 中二进制数转换为 16 进制数显示子程序
                MOV     CH, 4
ROTATE:         MOV     CL, 4
                ROL     BX, CL
```

```
                MOV     DL, BL
                AND     DL, 0FH
                ADD     DL, 30H
                CMP     DL, 3AH         ; 是 A~F 吗?
                JL      PRINT_IT
                ADD     DL, 07H
  PRINT_IT:     MOV     AH, 02H         ; 显示一个字符
                INT     21H
                DEC     CH
                JNZ     ROTATE
                RET
  BINIHEX       ENDP
CSEG            ENDS                    ; 以上定义代码段
; ---------------------------------------------------------------------
                END     START
```

5.23 已定义了两个整数变量 A 和 B，试编写程序完成下列功能：

(1) 若两个数中有一个是奇数，则将奇数存入 A 中，偶数存入 B 中；

(2) 若两个数中均为奇数，则将两数加 1 后存回原变量；

(3) 若两个数中均为偶数，则两个变量均不改变。

答：程序如下：

```
DSEG            SEGMENT
  A             DW  ?
  B             DW  ?
DSEG            ENDS
; ---------------------------------------------------------------------
CSEG            SEGMENT
  MAIN          PROC    FAR
                ASSUME  CS: CSEG, DS: DSEG
  START:        PUSH    DS              ; 设置返回 DOS
                SUB     AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX          ; 给 DS 赋值

  BEGIN:        MOV     AX, A
                MOV     BX, B
                XOR     AX, BX
                TEST    AX, 0001H       ; A 和 B 同为奇数或偶数吗?
                JZ      CLASS           ; A 和 B 都为奇数或偶数，转走
                TEST    BX, 0001H
                JZ      EXIT            ; B 为偶数，转走
                XCHG    BX, A           ; A 为偶数，将奇数存入 A 中
                MOV     B, BX           ; 将偶数存入 B 中
                JMP     EXIT
  CLASS:        TEST    BX, 0001H       ; A 和 B 都为奇数吗?
                JZ      EXIT            ; A 和 B 同为偶数，转走
                INC     B
                INC     A
  EXIT:         RET
  MAIN          ENDP
CSEG            ENDS                    ; 以上定义代码段
; ---------------------------------------------------------------------
                END     START
```

5.24 假设已编制好 5 个歌曲程序，它们的段地址和偏移地址存放在数据段的跳跃表 SINGLIST 中。试编制一程序，根据从键盘输入的歌曲编号 1~5，转去执行五个歌曲程序中的某一个。

答：程序如下：

```
DSEG        SEGMENT
  SINGLIST  DD   SING1
            DD   SING2
            DD   SING3
            DD   SING4
            DD   SING5
  ERRMSG    DB   'Error! Invalid parameter!', 0DH, 0AH, '$'
DSEG        ENDS
; ---------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC   FAR
            ASSUME  CS: CSEG, DS: DSEG
  START:    PUSH   DS                ; 设置返回 DOS
            SUB        AX, AX
            PUSH   AX
            MOV    AX, DSEG
            MOV    DS, AX            ; 给 DS 赋值

  BEGIN:    MOV    AH, 1             ; 从键盘输入的歌曲编号 1~5
            INT    21H
            CMP    AL, 0DH
            JZ     EXIT              ; 是回车符，则结束
            SUB        AL, '1'           ; 是 1~5 吗?
            JB     ERROR             ; 小于 1，错误
            CMP    AL, 4
            JA     ERROR             ; 大于 5，错误
            MOV    BX, OFFSET SINGLIST
            MUL    AX, 4             ; (AX)=(AL)*4,每个歌曲程序的首地址占 4 个字节
            ADD    BX, AX
            JMP    DWORD PTR[BX] ; 转去执行歌曲程序
  ERROR:    MOV    DX, OFFSET ERRMSG
            MOV    AH, 09H
            INT    21H               ; 显示错误信息
            JMP    BEGIN
  SING1:    ┆
            JMP    BEGIN
  SING2:    ┆
            JMP    BEGIN
  SING3:    ┆
            JMP    BEGIN
  SING4:    ┆
            JMP    BEGIN
  SING5:    ┆
            JMP    BEGIN
  EXIT:     RET
  MAIN      ENDP
CSEG        ENDS                     ; 以上定义代码段
; ---------------------------------------------------------------
            END    START
```

5.25 试用 8086 的乘法指令编制一个 32 位数和 16 位数相乘的程序；再用 80386 的乘法指令编制一个 32 位数和 16 位数相乘的程序，并定性比较两个程序的效率。

答：8086 的程序如下(假设为无符号数)：

```
DSEG        SEGMENT
  MUL1      DD   ?                  ; 32 位被乘数
  MUL2      DW   ?                  ; 16 位乘数
  MUL0      DW   0，0 ，0 ，0        ; 乘积用 64 位单元存放
```

```
DSEG        ENDS
; ---------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME  CS: CSEG, DS: DSEG
  START:    PUSH    DS              ; 设置返回 DOS
            SUB         AX, AX
            PUSH    AX
            MOV     AX, DSEG
            MOV     DS, AX          ; 给 DS 赋值

  BEGIN:    MOV     BX, MUL2        ; 取乘数
            MOV     AX, WORD PTR MUL1       ; 取被乘数低位字
            MUL     BX
            MOV     MUL0, AX        ; 保存部分积低位
            MOV     MUL0+2, DX      ; 保存部分积高位
            MOV     AX, WORD PTR[MUL1+2] ; 取被乘数高位字
            MUL     BX
            ADD     MUL0+2, AX      ; 部分积低位和原部分积高位相加
            ADC     MUL0+4, DX      ; 保存部分积最高位，并加上进位
  EXIT:     RET
  MAIN      ENDP
CSEG        ENDS                    ; 以上定义代码段
; ---------------------------------------------------------------
            END     START
```

80386 的程序如下(假设为无符号数)：

```
.386
DSEG        SEGMENT
  MUL1      DD  ?                   ; 32 位被乘数
  MUL2      DW  ?                   ; 16 位乘数
  MUL0      DD  0，0                ; 乘积用 64 位单元存放
DSEG        ENDS
; ---------------------------------------------------------------
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME  CS: CSEG, DS: DSEG
  START:    PUSH    DS              ; 设置返回 DOS
            SUB         AX, AX
            PUSH    AX
            MOV     AX, DSEG
            MOV     DS, AX          ; 给 DS 赋值

  BEGIN:    MOVZX   EBX, MUL2       ; 取乘数，并 0 扩展成 32 位
            MOV     EAX, MUL1       ; 取被乘数
            MUL     EBX
            MOV     DWORD PTR MUL0, EAX     ; 保存积的低位双字
            MOV     DWORD PTR[MUL0+4], EDX  ; 保存积的高位双字
  EXIT:     RET
  MAIN      ENDP
CSEG        ENDS                    ; 以上定义代码段
; ---------------------------------------------------------------
            END     START
```

80386 作 32 位乘法运算用一条指令即可完成，而 8086 则需用部分积作两次完成。

5.26 如数据段中在首地址为 MESS1 的数据区内存放着一个长度为 35 的字符串，要求把它们传送到附加段中的缓冲区 MESS2 中去。为提高程序执行效率，希望主要采用 MOVSD 指令来实现。试编写

这一程序。

答：80386 的程序如下：

```
            .386
            .MODEL SMALL
            .STACK  100H
            .DATA
MESS1   DB  '123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',?   ；长度为 35 的字符串
            .FARDATA
MESS2   DB  36 DUP (?)
            .CODE
START:      MOV     AX, @DATA
            MOV     DS, AX              ；给 DS 赋值
            MOV     AX, @FARDATA
            MOV     ES, AX              ；给 ES 赋值
            ASSUME  ES:@FARDATA

BEGIN:      LEA     ESI, MESS1
            LEA         EDI, MESS2
            CLD
            MOV     ECX, (35+1)/4           ；取传送的次数
            REP     MOVSD
；-------------------------------------------------------------
            MOV     AX, 4C00H           ；返回 DOS
            INT     21H
            END     START
```

5.27 试用比例变址寻址方式编写一 386 程序，要求把两个 64 位整数相加并保存结果。

答：80386 的程序如下：

```
            .386
            .MODEL SMALL
            .STACK  100H
            .DATA
DATA1   DQ  ?
DATA2   DQ  ?
            .CODE
START:      MOV     AX, @DATA
            MOV     DS, AX              ；给 DS 赋值

BEGIN:      MOV     ESI, 0
            MOV     EAX, DWORD  PTR  DATA2[ESI*4]
            ADD     DWORD  PTR  DATA1[ESI*4], EAX
            INC     ESI
            MOV     EAX, DWORD  PTR  DATA2[ESI*4]
            ADC     DWORD  PTR  DATA1[ESI*4], EAX
；-------------------------------------------------------------
            MOV     AX, 4C00H           ；返回 DOS
            INT     21H
            END     START
```

第六章. 习 题

6.1 下面的程序段有错吗？若有，请指出错误。

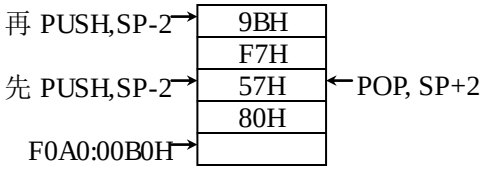
```
CRAY    PROC
        PUSH    AX
        ADD     AX, BX
        RET
ENDP    CRAY
```

答：程序有错。改正如下：

```
CRAY    PROC
        ADD     AX, BX
        RET
CRAY    ENDP                    ；CRAY 是过程名，应放在 ENDP 的前面
```

6.2 已知堆栈寄存器 SS 的内容是 0F0A0H，堆栈指示器 SP 的内容是 00B0H，先执行两条把 8057H 和 0F79BH 分别入栈的 PUSH 指令，然后执行一条 POP 指令。试画出示意图说明堆栈及 SP 内容的变化过程。

答：变化过程如右图所示：



6.2 题堆栈及 SP 内容的变化过程

6.3 分析下面的程序，画出堆栈最满时各单元的地址及内容。

```
; ********************************************
S_SEG   SEGMENT   AT   1000H    ；定义堆栈段
        DW        200 DUP (?)   ；200*2=190H
  TOS   LABEL     WORD
S_SEG   ENDS
; ********************************************
C_SEG   SEGMENT                 ；定义代码段
        ASSUME    CS: C_SEG, SS: S_SEG
 START:     MOV     AX, S_SEG
        MOV     SS, AX
        MOV     SP, OFFSET   TOS

        PUSH    DS
        MOV     AX, 0
        PUSH    AX
         ┇
        PUSH    T_ADDR
        PUSH    AX
        PUSHF
         ┇
        POPF
        POP     AX
        POP     T_ADDR
        RET
; ---------------------------------------------------------------
C_SEG   ENDS                    ；代码段结束
; ********************************************
        END     START           ；程序结束
```

6.3 题堆栈最满时各单元的地址及内容

答：堆栈最满时各单元的地址及内容如右图所示：

6.4 分析下面的程序，写出堆栈最满时各单元的地址及内容。

```
; ********************************************
STACK   SEGMENT   AT 500H       ；定义堆栈段
        DW        128 DUP (?)
  TOS   LABEL     WORD
STACK   ENDS
; ********************************************
CODE    SEGMENT                 ；定义代码段
MAIN    PROC    FAR             ；主程序部分
        ASSUME    CS: CODE, SS: STACK
 START:     MOV     AX, STACK
        MOV     SS, AX
        MOV     SP, OFFSET   TOS
        PUSH    DS
        SUB           AX, AX
        PUSH    AX
```

```
; MAIN PART OF PROGRAM GOES HERE
        MOV     AX, 4321H
        CALL    HTOA
        RET
MAIN    ENDP                    ; 主程序部分结束
; ------------------------------------------------------------
 HTOA PROC      NEAR            ; HTOA 子程序
        CMP     AX, 15
        JLE     B1
        PUSH    AX
        PUSH    BP
        MOV     BP, SP
        MOV     BX, [BP+2]
        AND     BX, 000FH
        MOV     [BP+2], BX
        POP     BP
        MOV     CL, 4
        SHR         AX, CL
        CALL    HTOA
        POP     BP
 B1:    ADD     AL, 30H
        CMP     AL, 3AH
        JL      PRINTIT
        ADD     AL, 7H
 PRINTIT: MOV   DL, AL
        MOV     AH, 2
        INT     21H
        RET
 HOTA   ENDP                    ; HOTA 子程序结束
; ------------------------------------------------------------
CODE    ENDS                    ; 代码段结束
; ******************************************
        END     START           ; 程序结束
```

0500:00EC→	
:00EE	返回 POP BP 地址
:00F0	0003H
:00F2	返回 POP BP 地址
:00F4	0002H
:00F6	返回 POP BP 地址
:00F8	0001H
:00FA	主程序返回地址
:00FC	0000
:00FE	(DS)
SP:	00EE

6.4 题堆栈最满时各单元的地址及内容

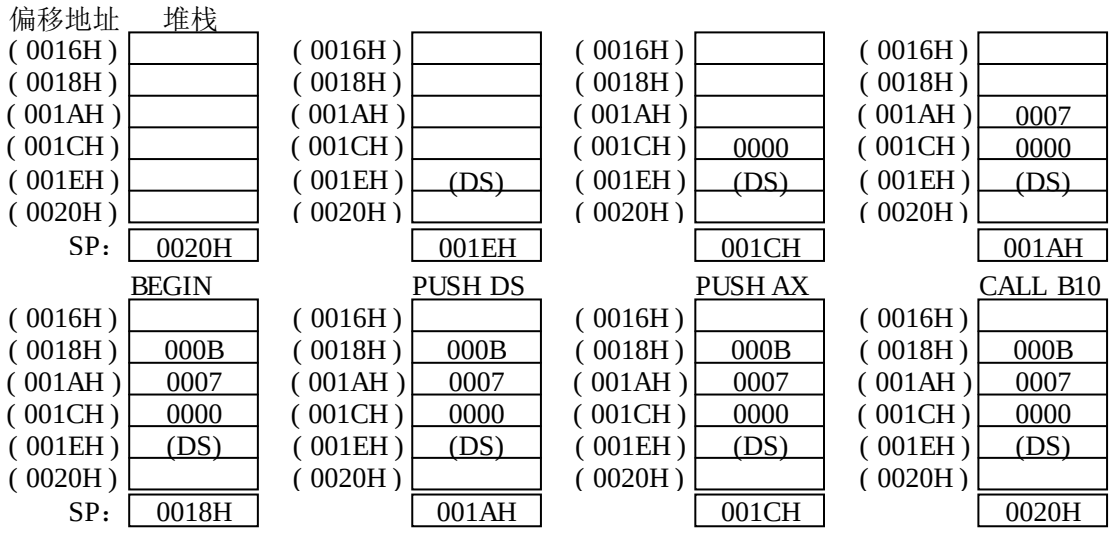
答：堆栈最满时各单元的地址及内容如右上图所示：

6.5 下面是一个程序清单，请在下面的图中填入此程序执行过程中的堆栈变化。

```
                        ; ***************************
0000                        STACKSG     SEGMENT
0000        20  [.                      DW    32 DUP (?)
                ????
                    ]
0040                        STACKSG     ENDS
                        ; ***************************
0000                        CODESG          SEGMENT    PARA   'CODE'
                        ; ---------------------------------
0000                        BEGIN       PROC    FAR
                                ASSUME    CS: CODESG, SS: STACKSG
0000        1E                          PUSH    DS
0001        2B  C0                      SUB     AX, AX
0003        50                          PUSH    AX
0004        E8  0008        R               CALL    B10
                        ; ---------------------------------
0007        CB                          RET
0008                        BEGIN       ENDP
                        ; ---------------------------------
0008                        B10         PROC
0008        E8  000C        R           CALL    C10
                        ; ---------------------------------
```

```
000B    C3                  RET
000C                B10     ENDP
                    ; --------------------------------
000C                C10     PROC
                    ; --------------------------------
000C    C3                  RET
000D                C10     ENDP
                    ; --------------------------------
000D                CODESG  ENDS
                    ; **************************
                            END     BEGIN
```

答：程序执行过程中的堆栈变化如下图所示。



6.6 写一段子程序 SKIPLINES，完成输出空行的功能。空出的行数在 AX 寄存器中。

答：程序如下：

```
CSEG        SEGMENT
  SKIPLINES PROC    FAR
            ASSUME  CS: CSEG
  BEGIN:    PUSH    CX
            PUSH    DX
            MOV     CX, AX
  DISP:     MOV     DL, 0DH         ；显示回车换行，即输出空行
            MOV     AH, 2           ；显示一个字符的 DOS 调用
            INT     21H
            MOV     DL, 0AH
            MOV     AH, 2           ；显示一个字符的 DOS 调用
            INT     21H
            LOOP    DISP
            POP     DX
            POP     CX
            RET
  SKIPLINES ENDP
            END
```

6.7 设有 10 个学生的成绩分别是 76，69，84，90，73，88，99，63，100 和 80 分。试编制一个子程序统计 60~69 分，70~79 分，80~89 分，90~99 分和 100 分的人数，分别存放到 S6，S7，S8，S9 和 S10 单元中。

答：程序如下：

```
DSEG        SEGMENT
```

```
  RECORD    DW   76，69，84，90，73，88，99，63，100，80
  S6        DW   0
  S7        DW   0
  S8        DW   0
  S9        DW   0
  S10       DW   0
DSEG        ENDS
; ****************************************
CSEG        SEGMENT
  MAIN      PROC   FAR
            ASSUME  CS: CSEG, DS: DSEG
  START:    PUSH   DS                ；设置返回 DOS
            SUB        AX, AX
            PUSH   AX
            MOV    AX, DSEG
            MOV    DS, AX            ；给 DS 赋值

  BEGIN:    MOV    CX, 10
            CALL   COUNT
              ┇                      ；后续程序
            RET
  MAIN      ENDP
; ------------------------------------------------------------
  COUNT         PROC   NEAR          ；成绩统计子程序
            MOV    SI, 0
  NEXT:     MOV    AX, RECORD[SI]
            MOV    BX, 10            ；以下 5 句是根据成绩计算相对 S6 的地址变化量
            DIV    BL                ；计算公式为：((成绩)/10-6)*2 送(BX)
            MOV    BL, AL            ；此时(BH)保持为 0 不变
            SUB        BX, 6             ；应为只统计 60 分以上成绩
            SAL    BX, 1             ；(BX)*2
            INC    S6[BX]            ；S6 是 S6，S7，S8，S9 和 S10 单元的首地址
            ADD    SI, 2
            LOOP   NEXT
            RET
  COUNT         ENDP                 ；COUNT 子程序结束
; ------------------------------------------------------------
CSEG        ENDS                     ；以上定义代码段
; ****************************************
            END    START
```

6.8 编写一个有主程序和子程序结构的程序模块。子程序的参数是一个 N 字节数组的首地址 TABLE，数 N 及字符 CHAR。要求在 N 字节数组中查找字符 CHAR，并记录该字符出现的次数。主程序则要求从键盘接收一串字符以建立字节数组 TABLE，并逐个显示从键盘输入的每个字符 CHAR 以及它在 TABLE 数组中出现的次数。(为简化起见，假设出现次数≤15，可以用 16 进制形式把它显示出来。)

答：程序如下：

```
DSEG        SEGMENT
  TABLE     DB   255 DUP (?)
  N         DW    255
  CHAR      DB   ?
  CHAR_N    DB   0                   ；用于记录 CHAR 出现的次数
  CRLF      DB   0DH, 0AH, '$'
DSEG        ENDS                     ；以上定义数据段
; ****************************************
STACK       SEGMENT
```

```
            DW    100 DUP (?)
  TOS       LABEL   WORD
STACK       ENDS                    ; 以上定义堆栈段
; ****************************************
CSEG        SEGMENT
  MAIN      PROC    FAR
            ASSUME  CS: CSEG, DS: DSEG, SS: STACK
  START:    MOV     AX, STACK
            MOV     SS, AX          ; 给 SS 赋值
            MOV     SP, OFFSET TOS  ; 给 SP 赋值
            PUSH    DS              ; 设置返回 DOS
            SUB     AX, AX
            PUSH    AX
            MOV     AX, DSEG
            MOV     DS, AX          ; 给 DS 赋值

  BEGIN:    MOV     BX, 0
            MOV     CX, 255         ; 最多输入 255 个字符
  INPUT:    MOV     AH, 1           ; 从键盘接收一个字符的 DOS 功能调用
            INT     21H
            CMP     AL, 0DH         ; 输入回车符结束输入
            JZ      IN_N
            MOV     TABLE [BX], AL
            INC     BX
            LOOP    INPUT
  IN_N:     MOV     N, BX           ; TABLE 数组中的字符个数送 N
            CALL    DISP_CRLF
  IN_CHAR:  MOV     AH, 1           ; 从键盘接收一个字符并回显的 DOS 功能调用
            INT     21H
            CMP     AL, 0DH         ; 输入回车符结束
            JZ      EXIT
            MOV     CHAR, AL        ; 输入的字符存入 CHAR 单元
            CALL    SEARCH          ; 调搜索字符子程序
            MOV     DL, ':'         ; 显示":"，在字符 CHAR(输入时回显)的后面
            MOV     AH, 2           ; 显示一个字符
            INT     21H
            MOV     DL, CHAR_N      ; 再显示 CHAR 出现的次数(次数≤15)
            AND     DL, 0FH
            ADD     DL, 30H
            CMP     DL, 39H
            JBE     NEXT
            ADD     DL, 07H         ; 是 A~F
  NEXT:     MOV     AH, 2           ; 显示一个字符
            INT     21H
            CALL    DISP_CRLF
            JMP     SHORT   IN_CHAR
  EXIT:     RET
  MAIN      ENDP
; ---------------------------------------------------------------
  SEARCH    PROC    NEAR            ; 搜索字符子程序
            MOV     SI, 0
            MOV     CX, N
            MOV     CHAR_N, 0
            MOV     AL, CHAR
  ROTATE:   CMP     AL, TABLE [SI]
            JNZ     ROTATE1
```

```
                INC     CHAR_N              ；搜索到字符，则出现次数+1
  ROTATE1:      INC     SI
                LOOP    ROTATE
                RET
  SEARCH        ENDP                        ；SEARCH 子程序结束
；-------------------------------------------------------------
  DISP_CRLF     PROC    NEAR                ；显示回车换行符子程序
                LEA         DX, CRLF
                MOV     AH, 09H
                INT     21H
                RET
  DISP_CRLF     ENDP                        ；DISP_CRLF 子程序结束
；-------------------------------------------------------------
  CSEG          ENDS                        ；以上定义代码段
；*****************************************
                END     START
```

6.9 编写一个子程序嵌套结构的程序模块，分别从键盘输入姓名及 8 个字符的电话号码，并以一定的格式显示出来。

主程序 TELIST：

- 显示提示符“INPUT NAME:”；
- 调用子程序 INPUT_NAME 输入姓名；
- 显示提示符“INPUT A TELEPHONE NUMBER:”；
- 调用子程序 INPHONE 输入电话号码；
- 调用子程序 PRINTLINE 显示姓名及电话号码。

子程序 INPUT_NAME：

- 调用键盘输入子程序 GETCHAR，把输入的姓名存放在 INBUF 缓冲区中；
- 把 INBUF 中的姓名移入输出行 OUTNAME。

子程序 INPHONE：

- 调用键盘输入子程序 GETCHAR，把输入的 8 位电话号码存放在 INBUF 缓冲区中；
- 把 INBUF 中的号码移入输出行 OUTPHONE。

子程序 PRINTLINE：

显示姓名及电话号码，格式为：

```
NAME      TEL.
X X X     XXXXXXXX
```

答：程序如下：

```
  DSEG          SEGMENT
    INBUF       DB   12 DUP (‘ ’)          ；输入缓冲区，初始值为空格
    OUTNAME     DB   16 DUP (‘ ’),         ；姓名输出行，初始值为空格
    OUTPHONE    DB   12 DUP (‘ ’), 0DH, 0AH, ‘$’；号码输出行，初始值为空格
    MESG1       DB   ‘INPUT  NAME：’, ‘$’
    MESG2       DB   ‘INPUT  A  TELEPHONE  NUMBER：’, ‘$’
    MESG3       DB   ‘NAME’, 12 DUP (‘ ’), ‘TEL.’, 0DH, 0AH, ‘$’
    CRLF        DB   0DH, 0AH, ‘$’
  DSEG          ENDS                       ；以上定义数据段
  ；*****************************************
  STACK         SEGMENT
                DW   100 DUP (?)
    TOS         LABEL   WORD
  STACK         ENDS                       ；以上定义堆栈段
  ；*****************************************
  CSEG          SEGMENT
    TELIST      PROC    FAR                ；主程序 TELIST
                ASSUME  CS: CSEG, DS: DSEG, ES: DSEG, SS: STACK
    START:      MOV     AX, STACK
```

```
                MOV     SS, AX          ; 给 SS 赋值
                MOV     SP, OFFSET TOS  ; 给 SP 赋值
                PUSH    DS              ; 设置返回 DOS
                SUB     AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX          ; 给 DS 赋值
                MOV     ES, AX          ; 给 ES 赋值

BEGIN:          LEA     DX, MESG1
                MOV     AH, 09H         ; 显示字符串功能调用
                INT     21H
                CALL    INPUT_NAME      ; 输入姓名
                LEA     DX, MESG2
                MOV     AH, 09H         ; 显示字符串功能调用
                INT     21H
                CALL    INPHONE         ; 输入电话号码
                CALL    PRINTLINE       ; 显示姓名及电话号码
                RET
TELIST          ENDP
; ---------------------------------------------------------------
INPUT_NAME      PROC    NEAR            ; 输入姓名子程序
                CALL    GETCHAR         ; 调输入字符子程序输入姓名
                LEA     SI, INBUF       ; 把 INBUF 中的姓名移入输出行 OUTNAME
                LEA     DI, OUTNAME
                MOV     CX, 12
                CLD
                REP     MOVSB
                RET
INPUT_NAME      ENDP                    ; INPUT_NAME 子程序结束
; ---------------------------------------------------------------
INPHONE         PROC    NEAR            ; 输入电话号码子程序
                CALL    GETCHAR         ; 调输入字符子程序输入电话号码
                LEA     SI, INBUF       ; 把 INBUF 中的电话号码移入输出行
                OUTPHONE
                LEA     DI, OUTPHONE
                MOV     CX, 12
                CLD
                REP     MOVSB
                RET
INPHONE         ENDP                    ; INPHONE 子程序结束
; ---------------------------------------------------------------
GETCHAR         PROC    NEAR            ; 键盘输入子程序
                MOV     AL, 20H         ; 先将 INBUF 中填满空格字符
                MOV     CX, 12
                LEA     DI, INBUF
                CLD
                REP     STOSB
                MOV     CX, 12          ; 向 INBUF 输入字符
                MOV     DI, 0
INPUT:          MOV     AH, 1           ; 从键盘接收一个字符并回显的 DOS 功能调用
                INT     21H
                CMP     AL, 0DH         ; 输入回车符返回
                JZ      QUIT
                MOV     INBUF[DI], AL
                INC     DI
```

```
                LOOP    INPUT
  QUIT:         CALL    DISP_CRLF
                RET
  GETCHAR       ENDP                    ; GETCHAR 子程序结束
; ---------------------------------------------------------------
  PRINTLINE PROC    NEAR                ; 显示姓名及电话号码子程序
                LEA     DX, MESG3
                MOV     AH, 09H         ; 显示字符串功能调用
                INT     21H
                LEA     DX, OUTNAME     ; 显示姓名及电话号码
                MOV     AH, 09H         ; 显示字符串功能调用
                INT     21H
                RET
  PRINTLINE ENDP                        ; PRINTLINE 子程序结束
; ---------------------------------------------------------------
  DISP_CRLF PROC    NEAR                ; 显示回车换行符子程序
                LEA     DX, CRLF
                MOV     AH, 09H
                INT     21H
                RET
  DISP_CRLF ENDP                        ; DISP_CRLF 子程序结束
; ---------------------------------------------------------------
CSEG            ENDS                    ; 以上定义代码段
; ***************************************
                END     START
```

6.10 编写子程序嵌套结构的程序，把整数分别用二进制和八进制形式显示出来。

主程序 BANDO：把整数字变量 VAL1 存入堆栈，并调用子程序 PAIRS；

子程序 PAIRS：从堆栈中取出 VAL1；调用二进制显示程序 OUTBIN 显示出与其等效的二进制数；输出 8 个空格；调用八进制显示程序 OUTOCT 显示出与其等效的八进制数；调用输出回车及换行符子程序。

答：程序如下：

```
DSEG            SEGMENT
  VAL1          DW    ?
  CRLF          DB    0DH, 0AH, '$'
DSEG            ENDS                    ; 以上定义数据段
; ***************************************
CSEG            SEGMENT
  BANDO         PROC    FAR             ; 主程序 BANDO
                ASSUME  CS: CSEG, DS: DSEG
  START:        PUSH    DS              ; 设置返回 DOS
                SUB     AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX          ; 给 DS 赋值

                PUSH    VAL1
                CALL    PAIRS
                RET
  BANDO         ENDP
; ---------------------------------------------------------------
  PAIRS         PROC    NEAR            ; PAIRS 子程序
                PUSH    BP
                MOV     BP, SP
                PUSH    BX
                MOV     BX, [BP+4]      ; 从堆栈中取出 VAL1
```

```
                CALL    OUTBIN          ；调用二进制显示子程序
                MOV     CX, 8           ；显示 8 个空格符
SPACE:          MOV     DL, ‘ ’
                MOV     AH, 2
                INT     21H
                LOOP    SPACE
                CALL    OUTOCT              ；调用八进制显示子程序
                CALL    DISP_CRLF
                POP     BX
                POP     BP
                RET     2
PAIRS           ENDP                    ；PAIRS 子程序结束
;-------------------------------------------------------------------
OUTBIN          PROC    NEAR            ；二进制显示子程序
                PUSH    BX
                MOV     CX, 16
ONEBIT:         ROL         BX, 1
                MOV     DX, BX
                AND     DX, 1
                OR      DL, 30H         ；转换为 ASCII 码
                MOV     AH, 2
                INT     21H
                LOOP    ONEBIT
                POP     BX
                RET
OUTBIN          ENDP                    ；OUTBIN 子程序结束
;-------------------------------------------------------------------
OUTOCT          PROC    NEAR            ；八进制显示子程序
                ROL         BX, 1           ；16 位二进制数包含 6 位八进制数,最高位仅 1
                位
                MOV     DX, BX
                AND     DX, 1
                OR      DL, 30H         ；转换为 ASCII 码
                MOV     AH, 2
                INT     21H
                MOV     CX, 5           ；余下还有 5 位八进制数
NEXT:           PUSH    CX
                MOV     CL, 3           ；1 位八进制数包含 3 位二进制数
                ROL         BX, CL
                MOV     DX, BX
                AND     DX, 07H
                OR      DL, 30H         ；转换为 ASCII 码
                MOV     AH, 2
                INT     21H
                POP     CX
                LOOP    NEXT
                RET
OUTOCT          ENDP                    ；OUTOCT 子程序结束
;-------------------------------------------------------------------
DISP_CRLF       PROC    NEAR            ；显示回车换行符子程序
                LEA         DX, CRLF
                MOV     AH, 09H
                INT     21H
                RET
DISP_CRLF       ENDP                    ；DISP_CRLF 子程序结束
;-------------------------------------------------------------------
```

```
CSEG        ENDS                    ；以上定义代码段
；*****************************************
            END     START
```

6.11 假定一个名为 MAINPRO 的程序要调用子程序 SUBPRO，试问：

(1) MAINPRO 中的什么指令告诉汇编程序 SUBPRO 是在外部定义的？

(2) SUBPRO 怎么知道 MAINPRO 要调用它？

答：(1) EXTRN SUBPRO:FAR

(2) PUBLIC SUBPRO

6.12 假定程序 MAINPRO 和 SUBPRO 不在同一模块中，MAINPRO 中定义字节变量 QTY 和字变量 VALUE 和 PRICE。SUBPRO 程序要把 VALUE 除以 QTY，并把商存在 PRICE 中。试问：

(1) MAINPRO 怎么告诉汇编程序外部子程序要调用这三个变量？

(2) SUBPRO 怎么告诉汇编程序这三个变量是在另一个汇编语言程序定义的？

答：(1) PUBLIC QTY, VALUE, PRICE

(2) EXTRN QTY:BYTE, VALUE:WORD, PRICE:WORD

6.13 假设：

(1) 在模块 1 中定义了双字变量 VAR1，首地址为 VAR2 的字节数据和 NEAR 标号 LAB1，它们将由模块 2 和模块 3 所使用；

(2) 在模块 2 中定义了字变量 VAR3 和 FAR 标号 LAB2，而模块 1 中要用到 VAR3，模块 3 中要用到 LAB2；

(3) 在模块 3 中定义了 FAR 标号 LAB3，而模块 2 中要用到它。

试对每个源模块给出必要的 EXTRN 和 PUBLIC 说明。

答：模块 1：

```
        EXTRN   VAR3: WORD
        PUBLIC  VAR1，VAR2，LAB1
```

模块 2：

```
        EXTRN   VAR1: DWORD，VAR2: BYTE，LAB1: NEAR，LAB3: FAR
        PUBLIC  VAR3，LAB2
```

模块 3：

```
        EXTRN   VAR1: DWORD，VAR2: BYTE，LAB1: NEAR，LAB2: FAR
        PUBLIC  LAB3
```

6.14 主程序 CALLMUL 定义堆栈段、数据段和代码段，并把段寄存器初始化，数据段中定义变量 QTY 和 PRICE；代码段中将 PRICE 装入 AX，QTY 装入 BX，然后调用子程序 SUBMUL。程序 SUBMUL 没有定义任何数据，它只简单地把 AX 中的内容(PRICE)乘以 BX 中的内容(QTY)，乘积放在 DX: AX 中。请编制这两个要连接起来的程序。

答：程序如下：

```
TITLE       CALLMUL                 ；主程序
EXTRN       SUBMUL: FAR
；-------------------------------------------------------
STACK       SEGMENT   PARA STACK 'STACK'
            DW   64 DUP (?)
  TOS       LABEL   WORD
STACK       ENDS
；-------------------------------------------------------
DATASG      SEGMENT   PARA 'DATA'
  QTY       DW   0140H
  PRICE     DW   2500H
DATASG      ENDS
；-------------------------------------------------------
CODESG      SEGMENT   PARA 'CODE'
  CALLMUL   PROC    FAR
            ASSUME   CS: CODESG, DS: DATASG, SS: STACK
  START:    MOV     AX, STACK
            MOV     SS, AX              ；给 SS 赋值
```

```
                MOV     SP, OFFSET TOS      ；给 SP 赋值
                PUSH    DS
                SUB         AX, AX
                POP     AX
                MOV     AX, DATASG
                MOV     DS, AX

                MOV     AX, PRICE
                MOV     BX, QTY
                CALL    SUBMUL
                RET
  CALLMUL   ENDP
CODESG      ENDS
; ------------------------------------------------------
                END     CALLMUL
; ************************************************************
TITLE       SUBMUL                      ；子程序
PUBLIC      SUBMUL
; ------------------------------------------------------
CODESG1     SEGMENT   PARA 'CODE'
                ASSUME  CS: CODESG1
  SUBMUL    PROC    FAR
                ASSUME  CS: CODESG1
                MUL     BX
                RET
  SUBMUL    ENDP
CODESG1     ENDS
; ------------------------------------------------------
                END
```

6.15 试编写一个执行以下计算的子程序 COMPUTE：

$$R \leftarrow X + Y - 3$$

其中 X，Y 及 R 均为字数组。假设 COMPUTE 与其调用程序都在同一代码段中，数据段 D_SEG 中包含 X 和 Y 数组，数据段 E_SEG 中包含 R 数组，同时写出主程序调用 COMPUTE 过程的部分。

如果主程序和 COMPUTE 在同一程序模块中，但不在同一代码段中，程序应如何修改？

如果主程序和 COMPUTE 不在同一程序模块中，程序应如何修改？

答：(1) 主程序和 COMPUTE 在同一代码段中的程序如下：

```
TITLE       ADDITION                    ；主程序
; ------------------------------------------------------
D_SEG       SEGMENT   PARA 'DATA'
  COUNT         EQU   10H
  X         DW    COUNT DUP (?)
  Y         DW    COUNT DUP (?)
D_SEG       ENDS
; ------------------------------------------------------
E_SEG       SEGMENT   PARA 'DATA'
  R         DW    COUNT DUP (?)
E_SEG       ENDS
; ------------------------------------------------------
C_SEG       SEGMENT   PARA 'CODE'
  ADDITION  PROC    FAR
                ASSUME  CS: C_SEG, DS: D_SEG, ES: E_SEG
  START:    PUSH    DS
                SUB         AX, AX
                PUSH    AX
                MOV     AX, D_SEG
                MOV     DS, AX
                MOV     AX, E_SEG
```

```
                MOV     ES, AX
                CALL    COMPUTE             ；调用求和子程序
                RET
  ADDITION      ENDP
；*********************************************
  COMPUTE       PROC    NEAR                ；同一段的求和子程序
                MOV     CX, COUNT
                MOV     BX, 0
  REPEAT:       MOV     AX, X[BX]
                ADD     AX, Y[BX]
                SUB     AX, 3
                MOV     ES: R[BX], AX
                RET
  COMPUTE       ENDP
；---------------------------------------------------
C_SEG           ENDS
；*********************************************
                END     START
```

(2) 主程序和 COMPUTE 在同一程序模块中，但不在同一代码段中的程序如下：

```
TITLE           ADDITION                    ；主程序
；---------------------------------------------------
D_SEG           SEGMENT   PARA ‘DATA’
  COUNT           EQU   10H
  X             DW   COUNT DUP (?)
  Y             DW   COUNT DUP (?)
D_SEG           ENDS
；---------------------------------------------------
E_SEG           SEGMENT   PARA ‘DATA’
  R             DW   COUNT DUP (?)
E_SEG           ENDS
；---------------------------------------------------
C_SEG           SEGMENT   PARA ‘CODE’
  ADDITION      PROC    FAR
                ASSUME   CS: C_SEG, DS: D_SEG, ES: E_SEG
  START:        PUSH    DS
                SUB     AX, AX
                POP     AX
                MOV     AX, D_SEG
                MOV     DS, AX
                MOV     AX, E_SEG
                MOV     ES, AX
                CALL    FAR PTR COMPUTE     ；调用求和子程序
                RET
  ADDITION      ENDP
C_SEG           ENDS
；*********************************************
CODESG          SEGMENT   PARA ‘CODE’
                ASSUME   CS: CODESG
  COMPUTE       PROC    FAR                 ；不同段的求和子程序
                MOV     CX, COUNT
                MOV     BX, 0
  REPEAT:       MOV     AX, X[BX]
                ADD     AX, Y[BX]
                SUB     AX, 3
                MOV     ES: R[BX], AX
                RET
  COMPUTE       ENDP
```

```
; ------------------------------------------------------
CODESG      ENDS
; ******************************************
            END         START
```

(3) 主程序和 COMPUTE 不在同一程序模块中的程序如下：

```
TITLE       ADDITION                    ; 主程序
EXTRN       COMPUTE: FAR
PUBLIC      COUNT, X, Y, R
; ------------------------------------------------------
D_SEG       SEGMENT   PARA 'DATA'
  COUNT         DW    10H
  X             DW    10H DUP (?)
  Y             DW    10H DUP (?)
D_SEG       ENDS
; ------------------------------------------------------
E_SEG       SEGMENT   PARA 'DATA'
  R             DW    10H DUP (?)
E_SEG       ENDS
; ------------------------------------------------------
C_SEG       SEGMENT   PARA 'CODE'
  ADDITION  PROC      FAR
            ASSUME    CS: C_SEG, DS: D_SEG, ES: E_SEG
  START:    PUSH      DS
            SUB             AX, AX
            POP       AX
            MOV       AX, D_SEG
            MOV       DS, AX
            MOV       AX, E_SEG
            MOV       ES, AX
            CALL      FAR PTR COMPUTE   ; 调用求和子程序
            RET
  ADDITION  ENDP
C_SEG       ENDS
; ------------------------------------------------------
            END         START
; ************************************************************

TITLE       COMPUTE                     ; 求和子程序
EXTRN       COUNT:WORD, X:WORD, Y:WORD, R:WORD
PUBLIC      COMPUTE
; ------------------------------------------------------
CODESG      SEGMENT   PARA 'CODE'
            ASSUME    CS: CODESG
  COMPUTE   PROC      FAR               ; 不同模块的求和子程序
            MOV       CX, COUNT
            MOV       BX, 0
  REPEAT:   MOV       AX, X[BX]
            ADD       AX, Y[BX]
            SUB             AX, 3
            MOV       ES: R[BX], AX
            RET
  COMPUTE   ENDP
; ------------------------------------------------------
CODESG      ENDS
; ******************************************
            END
```

第七章. 习 题

7.1 编写一条宏指令 CLRB，完成用空格符将一字符区中的字符取代的工作。字符区首地址及其长度为变元。

答：宏定义如下：

```
CLRB    MACRO  N, CFIL
        MOV    CX, N
        CLD
        MOV    AL, ' '          ;; 取空格符的 ASCII 码
        LEA    DI, CFIL
        REP    STOSB
        ENDM
```

7.2 某工厂计算周工资的方法是每小时的工资率 RATE 乘以工作时间 HOUR，另外每工作满 10 小时加奖金 3 元，工资总数存放在 WAG 中。请将周工资的计算编写成一条宏指令 WAGES，并展开宏调用：

```
WAGES  R1, 42, SUM
```

答：宏定义如下：

```
WAGES   MACRO  RATE, HOUR, WAG
        MOV    AL, HOUR         ;; 计算周工资(WAG)，公式为：HOUR* RATE
        MOV    BL, RATE
        MUL    BL
        MOV    WAG, AX
        MOV    AL, HOUR         ;; 计算奖金存入(AX)，公式为：HOUR/10 的商*3
        MOV    AH, 0
        MOV    BL, 10
        DIV    BL
        MOV    BL, 3
        MUL    BL
        ADD    WAG, AX              ;; 计算周工资总数
        ENDM
```

宏调用：

```
        WAGES  R1, 42, SUM
```

宏展开：

```
1       MOV    AL, 42
1       MOV    BL, R1
1       MUL    BL
1       MOV    SUM, AX
1       MOV    AL, 42
1       MOV    AH, 0
1       MOV    BL, 10
1       DIV    BL
1       MOV    BL, 3
1       MUL    BL
1       ADD    SUM, AX
```

7.3 给定宏定义如下：(注意：此宏指令的功能是 V3←|V1-V2|)

```
DIF     MACRO  X, Y
        MOV    AX, X
        SUB    AX, Y
        ENDM
ABSDIF  MACRO  V1, V2, V3
        LOCAL  CONT
        PUSH   AX
        DIF    V1, V2
        CMP    AX, 0
        JGE    CONT
        NEG    AX
```

```
CONT:   MOV     V3, AX
        POP     AX
        ENDM
```

试展开以下调用，并判定调用是否有效。

(1) ABSDIF P1, P2, DISTANCE
(2) ABSDIF [BX], [SI], X[DI], CX
(3) ABSDIF [BX][SI], X[BX][SI], 240H
(4) ABSDIF AX, AX, AX

答：(1) 宏调用 ABSDIF P1, P2, DISTANCE 的宏展开如下：此宏调用有效。

```
1               PUSH    AX
1               DIF     P1, P2
1               MOV     AX, P1
1               SUB     AX, P2
1               CMP     AX, 0
1               JGE     ??0000
1               NEG     AX
1       ??0000: MOV     DISTANCE, AX
1               POP     AX
```

(2) 宏调用 ABSDIF [BX], [SI], X[DI], CX 的宏展开如下：此宏调用有效。

```
1               PUSH    AX
1               DIF     [BX], [SI]
1               MOV     AX, [BX]
1               SUB     AX, [SI]
1               CMP     AX, 0
1               JGE     ??0001
1               NEG     AX
1       ??0001: MOV     X[DI], AX
1               POP     AX
```

(3) 宏调用 ABSDIF [BX][SI], X[BX][SI], 240H 的宏展开如下：此宏调用无效。

```
1               PUSH    AX
1               DIF     [BX][SI], X[BX][SI]
1               MOV     AX, [BX][SI]
1               SUB     AX, X[BX][SI]
1               CMP     AX, 0
1               JGE     ??0002
1               NEG     AX
1       ??0002: MOV     240H, AX
1               POP     AX
```

(4) 宏调用 ABSDIF AX, AX, AX 的宏展开如下：此宏调用有效但无多大意义。

```
1               PUSH    AX
1               DIF     AX, AX
1               MOV     AX, AX
1               SUB     AX, AX
1               CMP     AX, 0
1               JGE     ??0003
1               NEG     AX
1       ??0003: MOV     AX, AX
1               POP     AX
```

7.4 试编制宏定义，要求把存储器中的一个用 EOT（ASCII 码 04H）字符结尾的字符串传送到另一个存储区去。

答：宏定义如下：

```
SEND        MACRO  SCHARS, DCHARS
            LOCAL  NEXT, EXIT
            PUSH   AX
            PUSH   SI
            MOV    SI, 0
 NEXT:      MOV    AL, SCHARS[SI]
            MOV    DCHARS[SI], AL
```

```
                CMP     AL, 04H          ;; 是 EOT 字符吗?
                JZ      EXIT
                INC     SI
                JMP     NEXT
EXIT:           POP     SI
                POP     AX
                ENDM
```

7.5 宏指令 BIN_SUB 完成多个字节数据连减的功能：

RESULT←(A-B-C-D-…)

要相减的字节数据顺序存放在首地址为 OPERAND 的数据区中，减数的个数存放在 COUNT 单元中，最后结果存入 RESULT 单元。请编写此宏指令。

答：宏定义如下：

```
BIN_SUB         MACRO  RESULT, A, OPERAND, COUNT
                LOCAL  NEXT_SUB
                PUSH   CX
                PUSH   BX
                PUSH   AX
                MOV    CX, COUNT
                MOV    AL, A
                LEA        BX, OPERAND
                CLC
NEXT_SUB:       SBB    AL, [BX]
                INC    BX
                LOOP   NEXT_SUB
                MOV    RESULT, AL
                POP    AX
                POP    BX
                POP    CX
                ENDM
```

7.6 请用宏指令定义一个可显示字符串 GOOD: ‘GOOD STUDENTS: CLASSX NAME’，其中 X 和 NAME 在宏调用时给出。

答：宏定义如下：

```
DISP_GOOD  MACRO  X, NAME
           GOOD   DB  ‘GOOD STUDENTS: CLASS&X &NAME’, 0DH, 0AH, ‘$’
           ENDM
```

7.7 下面的宏指令 CNT 和 INC1 完成相继字存储。

```
CNT             MACRO  A, B
          A&B   DW   ?
          ENDM
INC1      MACRO  A, B
          CNT        A, %B
 B=B+1
          ENDM
```

请展开下列宏调用：

```
C=0
          INC1   DATA, C
          INC1   DATA, C
```

答：宏展开如下：

```
      C=0
                 INC1   DATA, C
      1              DATA0  DW   ?
                 INC1   DATA, C
      1              DATA0  DW   ?          （注意：C 为 0 没有变）
```

7.8 定义宏指令并展开宏调用。宏指令 JOE 把一串信息‘MESSAGE NO. K’存入数据存储区 XK 中。宏调用为：

```
I=0
        JOE     TEXT, I
          ┆
        JOE     TEXT, I
          ┆
        JOE     TEXT, I
          ┆
```

答：宏定义如下：

```
MARY    MACRO  X, K
        X&K    DB   'MESSAGE NO. &K'
        ENDM
JOE     MACRO  A, I
        MARY   A, %I
 I=I+1
        ENDM
```

宏调用和宏展开：

```
   I=0
        JOE     TEXT, I
1           TEXT0   DB   'MESSAGE NO. 0'
             ┆
        JOE     TEXT, I
1           TEXT1   DB   'MESSAGE NO. 1'
             ┆
        JOE     TEXT, I
1           TEXT2   DB   'MESSAGE NO. 2'
```

7.9 宏指令 STORE 定义如下：

```
STORE   MACRO  X, N
        MOV    X+I, I
 I=I+1
        IF     I-N
        STORE  X, N
        ENDIF
        ENDM
```

试展开下列宏调用：

```
I=0
        STORE  TAB, 7
```

答：宏展开如下：

```
   I=0
        STORE  TAB, 7
1           MOV    TAB+0, 0
1           MOV    TAB+1, 1
1           MOV    TAB+2, 2
1           MOV    TAB+3, 3
1           MOV    TAB+4, 4
1           MOV    TAB+5, 5
1           MOV    TAB+6, 6
```

7.10 试编写非递归的宏指令，使其完成的工作与 7.9 题的 STORE 相同。

答：宏定义如下：

```
STORE   MACRO  K
        MOV    TAB+K, K
        ENDM
```

宏调用：

```
   I=0
        REPT   7
        STORE  %I
   I=I+1
        ENDM
```

7.11 试编写一段程序完成以下功能，如给定名为 X 的字符串长度大于 5 时，下列指令将汇编 10 次。

```
ADD     AX, AX
```

答：程序段如下：

```
X       DB  'ABCDEFG'
        IF      ($-X)  GT  5
          REPT  10
                ADD     AX, AX
          ENDM
        ENDIF
```

7.12 定义宏指令 FINSUM：比较两个数 X 和 Y(X、Y 为数，而不是地址)，若 X>Y 则执行 SUM←X+2*Y；否则执行 SUM←2*X+Y。

答：宏定义如下：

```
CALCULATE  MACRO  A, B, RESULT          ;; 计算 RESULT←2*A+B
           MOV    AX, A
           SHL    AX, 1
           ADD    AX, B
           MOV    RESULT, AX
           ENDM
FINSUM     MACRO  X, Y, SUM
           IF     X  GT  Y
                  CALCULATE   Y, X, SUM
           ELSE
                  CALCULATE   X, Y, SUM
           ENDIF
           ENDM
```

7.13 试编写一段程序完成以下功能：如变元 X='VT55'，则汇编 MOV TERMINAL, 0；否则汇编 MOV TERMINAL, 1。

答：宏定义如下：

```
BRANCH      MACRO  X
            IFIDN  <X>, <VT55>
                   MOV  TERMINAL, 0
            ELSE
                   MOV  TERMINAL, 1
            ENDIF
            ENDM
```

7.14 对于 DOS 功能调用，所有的功能调用都需要在 AH 寄存器中存放功能码，而其中有一些功能需要在 DX 中放一个值。试定义宏指令 DOS21，要求只有在程序中定义了缓冲区时，汇编为：

```
MOV    AH, DOSFUNC
MOV    DX, OFFSET  BUFF
INT    21H
```

否则，无 MOV DX, OFFSET BUFF 指令。并展开以下宏调用：

```
DOS21  01
DOS21  0AH, IPFIELD
```

答：宏定义如下：

```
DOS21      MACRO  DOSFUNC, BUFF
           MOV    AH, DOSFUNC
           IFDEF  BUFF
                  MOV    DX, OFFSET  BUFF
           ENDIF
           INT    21H
           ENDM
```

宏展开：

```
           DOS21  01
  1             MOV    AH, 01
  1             INT    21H
           DOS21  0AH, IPFIELD
```

```
1               MOV     AH, 0AH
1               MOV     DX, OFFSET   IPFIELD
1               INT     21H
```

7.15 编写一段程序，使汇编程序根据 SIGN 中的内容分别产生不同的指令。如果(SIGN)=0，则用字节变量 DIVD 中的无符号数除以字节变量 SCALE；如果(SIGN)=1，则用字节变量 DIVD 中的带符号数除以字节变量 SCALE，结果都存放在字节变量 RESULT 中。

答：程序段如下：

```
                MOV     AL, DIVD
        IF      SIGN
                MOV     AH, 0
                DIV     SCALE
        ELSE
                CBW
                IDIV    SCALE
        ENDIF
                MOV     RESULT, AL
```

7.16 试编写宏定义 SUMMING，要求求出双字数组中所有元素之和，并把结果保存下来。该宏定义的哑元应为数组首址 ARRAY，数组长度 COUNT 和结果存放单元 RESULT。

答：宏定义如下：

```
SUMMING     MACRO ARRAY，COUNT，RESULT
            LOCAL ADDITION
            MOV   ESI, 0
            MOV   ECX, COUNT
ADDITION:   MOV   EAX, ARRAY[ESI*4]   ;；双字为 4 字节
            ADD   RESULT, EAX
            ADC   RESULT+4, 0         ;；将进位加到结果的高位双字中
            INC   ESI
            LOOP  ADDITION
            ENDM
```

7.17 为下列数据段中的数组编制一程序，调用题 7.16 的宏定义 SUMMING，求出该数组中各元素之和。

```
DATA    DD  101246，274365，843250，475536
SUM         DQ  ?
```

答：程序如下：

```
SUMMING     MACRO ARRAY，COUNT，RESULT
            LOCAL ADDITION
            MOV   ESI, 0
            MOV   ECX, COUNT
ADDITION:   MOV   EAX, ARRAY[ESI*4]   ;；双字为 4 字节
            ADD   RESULT, EAX
            ADC   RESULT+4, 0         ;；将进位加到结果的高位双字中
            INC   ESI
            LOOP  ADDITION
            ENDM

            .MODEL SMALL
            .386
            .DATA
DATA    DD  101246，274365，843250，475536
SUM     DQ  ?
            .CODE
START：     MOV   AX, @DATA
            MOV   DS, AX
            SUMMING   DATA, 4, SUM
            MOV   AX, 4C00H
            INT   21H
```

```
                END     START
```

7.18 如把题 7.16 中的宏定义存放在一个宏库中，则题 7.17 的程序应如何修改？

答：程序修改如下：

```
                INCLUDE     MACRO.MAC       ；假设存放的宏库名为 MACRO.MAC
                .MODEL SMALL
                .386
                .DATA
        DATA    DD   101246，274365，843250，475536
        SUM     DQ   ?
                .CODE
        START:  MOV     AX, @DATA
                MOV     DS, AX
                SUMMING   DATA, 4, SUM
                MOV     AX, 4C00H
                INT     21H
                END     START
```

第八章. 习 题

8.1 写出分配给下列中断类型号在中断向量表中的物理地址。

(1) INT 12H (2) INT 8

答：(1) 中断类型号 12H 在中断向量表中的物理地址为 00048H、00049H、0004AH、0004BH;

(2) 中断类型号 8 在中断向量表中的物理地址为 00020H、00021H、00022H、00023H。

8.2 用 CALL 指令来模拟实现 INT 21H 显示字符 T 的功能。

答：

```
MOV     AH, 2
MOV     DL, 'T'
PUSH    DS
PUSHF                           ；因中断服务程序的返回指令是 IRET，而不是 RET
MOV     BX, 0
MOV     DS, BX
CALL    DWORD PTR[21H*4]        ；用 CALL 指令调用 21H 的中断服务程序
POP     DS
```

8.3 写出指令将一个字节数据输出到端口 25H。

答：指令为：OUT 25H, AL

8.4 写出指令将一个字数据从端口 1000H 输入。

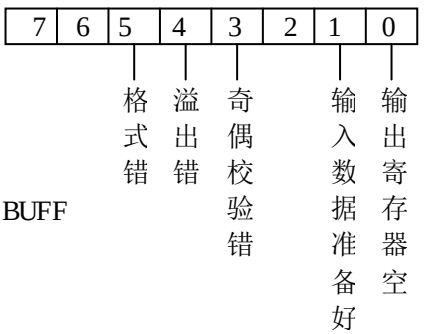
答：指令为：

```
        MOV     DX, 1000H
        IN      AX, DX
```

8.5 假定串行通讯口的输入数据寄存器的端口地址为 50H，状态寄存器的端口地址为 51H，状态寄存器各位为 1 时含义如右图所示，请编写一程序：输入一串字符并存入缓冲区 BUFF，同时检验输入的正确性，如有错则转出错处理程序 ERROR_OUT。

答：程序段如下：

```
        MOV     DI, 0
        MOV     CX, 80          ；最多输入 80 个字符
BEGIN:  IN      AL, 51H         ；查询输入是否准备好？
        TEST    AL, 02H
        JZ      BEGIN
        IN      AL, 50H         ；输入数据并存入缓冲区 BUFF
        MOV     BUFF[DI], AL
        INC     DI
        IN      AL, 51H         ；判断是否有错？
        TEST    AL, 00111000B
        JNZ     ERROR_OUT
```



8.3 状态寄存器各位含义

```
        LOOP    BEGIN
          ┆
```

8.6 试编写程序，它轮流测试两个设备的状态寄存器，只要一个状态寄存器的第 0 位为 1，则就与其相应的设备输入一个字符；如果其中任一状态寄存器的第 3 位为 1，则整个输入过程结束。两个状态寄存器的端口地址分别是 0024H 和 0036H，与其相应的数据输入寄存器的端口地址则为 0026H 和 0038H，输入字符分别存入首地址为 BUFF1 和 BUFF2 的存储区中。

答：程序段如下：

```
                MOV     DI, 0
                MOV     SI, 0
BEGIN:          IN      AL, 24H
                TEST    AL, 08H         ；查询第一个设备的输入是否结束？
                JNZ     EXIT
                TEST    AL, 01H         ；查询第一个设备的输入是否准备好？
                JZ      BEGIN1
                IN      AL, 26H         ；输入数据并存入缓冲区 BUFF1
                MOV     BUFF1[DI], AL
                INC     DI
BEGIN1:         IN      AL, 36H
                TEST    AL, 08H         ；查询第二个设备的输入是否结束
                JNZ     EXIT
                TEST    AL, 01H         ；查询第二个设备的输入是否准备好？
                JZ      BEGIN
                IN      AL, 38H         ；输入数据并存入缓冲区 BUFF2
                MOV     BUFF2[SI], AL
                INC     SI
                JMP     BEGIN
EXIT:           ┆
```

8.7 假定外部设备有一台硬币兑换器，其状态寄存器的端口地址为 0006H，数据输入寄存器的端口地址为 0005H，数据输出寄存器的端口地址为 0007H。试用查询方式编制一程序，该程序作空闲循环等待纸币输入，当状态寄存器第 2 位为 1 时，表示有纸币输入，此时可从数据输入寄存器输入的代码中测出纸币的品种，一角纸币的代码为 01，二角纸币为 02，五角纸币则为 03。然后程序在等待状态寄存器的第 3 位变为 1 后，把应兑换的五分硬币数(用 16 进制表示)从数据输出寄存器输出。

答：程序段如下：

```
BEGIN:          IN      AL, 06H         ；查询是否有纸币输入？
                TEST    AL, 04H
                JZ      BEGIN
                IN      AL, 05H         ；测试纸币的品种
                CMP     AL, 01H         ；是一角纸币吗？
                JNE     NEXT1
                MOV     AH, 02          ；是一角纸币，输出 2 个 5 分硬币
                JMP     NEXT
NEXT1:          CMP     AL, 02H         ；是二角纸币吗？
                JNE     NEXT2
                MOV     AH, 04          ；是二角纸币，输出 4 个 5 分硬币
                JMP     NEXT
NEXT2:          CMP     AL, 03H         ；是五角纸币吗？
                JNE     BEGIN
                MOV     AH, 10          ；是五角纸币，输出 10 个 5 分硬币
NEXT:           IN      AL, 06H         ；查询是否允许输出 5 分硬币？
                TEST    AL, 08H
                JZ      NEXT
                MOV     AL, AH          ；输出 5 分硬币
                OUT     07H, AL
```

```
        JMP     BEGIN
```

8.8 给定(SP)=0100H，(SS)=0300H，(FLAGS)=0240H，以下存储单元的内容为(00020)=0040H，(00022)=0100H，在段地址为 0900 及偏移地址为 00A0H 的单元中有一条中断指令 INT 8，试问执行 INT 8 指令后，SP，SS，IP，FLAGS 的内容是什么？栈顶的三个字是什么？

答：执行 INT 8 指令后，(SP)=00FAH，(SS)=0300H，(CS)=0100H，(IP)=0040H，(FLAGS)=0040H
栈顶的三个字是：原(IP)=00A2H，原(CS)=0900H，原(FLAGS)=0240H

8.9 类型 14H 的中断向量在存储器的哪些单元里？

答：在 0000:0050H，0000:0051H，0000:0052H，0000:0053H 四个字节中。

8.10 假定中断类型 9H 的中断处理程序的首地址为 INT_ROUT，试写出主程序中为建立这一中断向量而编制的程序段。

答：程序段如下：

```
        ┇
MOV     AL, 1CH         ；取原中断向量，并保护起来
MOV     AH, 35H
INT     21H
PUSH    ES
PUSH    BX
PUSH    DS
MOV     AX, SEG INT_ROUT
MOV     DS, AX
MOV     DX, OFFSET INT_ROUT
MOV     AL, 09H
MOV     AH, 25H         ；设置中断向量功能调用
INT     21H
POP     DS
        ┇
POP     DX              ；还原原中断向量
POP     DS
MOV     AL, 1CH
MOV     AH, 25H
INT     21H
```

8.11 编写指令序列，使类型 1CH 的中断向量指向中断处理程序 SHOW_CLOCK。

答：程序段如下：

```
        ┇
MOV     AL, 1CH
MOV     AH, 35H         ；取中断向量功能调用，取原中断向量
INT     21H
PUSH    ES
PUSH    BX
PUSH    DS
MOV     AX, SEG SHOW_CLOCK
MOV     DS, AX
MOV     DX, OFFSET SHOW_CLOCK
MOV     AL, 1CH
MOV     AH, 25H         ；设置中断向量功能调用
INT     21H
POP     DS
        ┇
POP     DX
POP     DS
MOV     AL, 1CH
MOV     AH, 25H         ；设置中断向量功能调用，还原原中断向量
INT     21H
```

⋮

8.12 如设备 D1，D2，D3，D4，D5 是按优先级次序排列的，设备 D1 的优先级最高。而中断请求的次序如下所示，试给出各设备的中断处理程序的运行次序。假设所有的中断处理程序开始后就有 STI 指令。

(1) 设备 D3 和 D4 同时发出中断请求。
(2) 在设备 D3 的中断处理程序完成之前，设备 D2 发出中断请求。
(3) 在设备 D4 的中断处理程序未发出中断结束命令(EOI)之前，设备 D5 发出中断请求。
(4) 以上所有中断处理程序完成并返回主程序，设备 D1，D3，D5 同时发出中断请求。

答：各设备的中断处理程序的运行次序是：INT_D3，INT_D2 嵌套 INT_D3，INT_D4，INT_D5；INT_D1，INT_D3，INT_D5。

8.13 在 8.12 题中假设所有的中断处理程序中都没有 STI 指令，而它们的 IRET 指令都可以由于 FLAGS 出栈而使 IF 置 1，则各设备的中断处理程序的运行次序应是怎样的？

答：各设备的中断处理程序的运行次序是：INT_D3，INT_D2，INT_D4，INT_D5；INT_D1，INT_D3，INT_D5。

8.14 试编制一程序，要求测出任一程序的运行时间，并把结果打印出来。

答：程序段如下：

```
TITLE       TEST_TIME.EXE           ; 测试程序运行时间程序
; ****************************************
DSEG        SEGMENT                 ; 定义数据段
  COUNT         DW   0                  ; 记录系统时钟(18.2 次中断/秒)的中断次数
  SEC       DW   0                  ; 存放秒钟数
  MIN       DW   0                  ; 存放分钟数
  HOURS     DW   0                  ; 存放小时数
  PRINTTIME DB   0DH, 0AH, 'The time of  exection program is:'
  CHAR_NO   EQU  $- PRINTTIME
DSEG        ENDS                    ; 以上定义数据段
; ****************************************
CSEG        SEGMENT                 ; 定义代码段
  MAIN      PROC   FAR
            ASSUME  CS: CSEG, DS: DSEG
  START:    PUSH   DS               ; 设置返回 DOS
            SUB        AX, AX
            PUSH   AX
            MOV    AX, DSEG
            MOV    DS, AX           ; 给 DS 赋值

            MOV    AL, 1CH          ; 取原来的 1CH 中断向量
            MOV    AH, 35H
            INT    21H
            PUSH   ES               ; 保存原来的 1CH 中断向量
            PUSH   BX

            PUSH   DS               ; 设置新的 1CH 中断向量
            MOV    AX, SEG CLINT
            MOV    DS, AX
            MOV    DX, OFFSET CLINT
            MOV    AL, 1CH
            MOV    AH, 25H
            INT    21H
            POP    DS

            IN     AL, 21H          ; 清除时间中断屏蔽位并开中断
            AND    AL, 0FEH
            OUT    21H, AL
```

```
                STI
                 ┆                          ；要求测试时间的程序段
                POP     DX              ；恢复原来的 1CH 中断向量
                POP     DS
                MOV     AL, 1CH
                MOV     AH, 25H
                INT     21H
                CALL    PRINT           ；打印输出测试时间
                RET                     ；返回 DOS
MAIN            ENDP
; ---------------------------------------------------------------------
CLINT           PROC    NEAR            ；中断服务子程序
                PUSH    DS
                PUSH    BX
                MOV     BX, SEG COUNT
                MOV     DS, BX
                LEA     BX, COUNT
                INC     WORD PTR [BX]   ；记录系统时钟的中断次数单元+1
                CMP     WORD PTR [BX],18    ；有 1 秒钟吗？
                JNE     TIMEOK
                CALL    INCTEST         ；有 1 秒钟，转去修改时间
ADJ:            CMP     HOURS, 12       ；有 12 小时吗？
                JLE     TIMEOK
                SUB     HOURS, 12       ；有 12 小时，将小时数减去 12
TIMEOK:         MOV     AL, 20H         ；发中断结束命令
                OUT     20H, AL
                POP     BX
                POP     DS
                IRET
CLINT           ENDP                    ；CLINT 中断服务子程序结束
; ---------------------------------------------------------------------
INCTEST         PROC    NEAR            ；修改时间子程序
                MOV     WORD PTR [BX], 0 ；中断次数单元或秒单元或分单元清 0
                ADD     BX, 2
                INC     WORD PTR [BX]   ；秒单元或分单元或时单元+1
                CMP     WORD PTR [BX],60 ；有 60 秒或 60 分吗？
                JLE     RETURN
                CALL    INCTEST         ；先修改秒单元，再修改分单元，再修改时单元
RETURN:         RET
INCTEST         ENDP                    ；INCTEST 子程序结束
; ---------------------------------------------------------------------
PRINT           PROC    NEAR            ；打印输出子程序
                LEA     BX, PRINTTIME   ；打印输出 PRINTTIME 信息
                MOV     CX, CHAR_NO
ROTATE:         MOV     DL, [BX]
                MOV     AH, 05H
                INT     21H
                INC     BX
                LOOP    ROTATE
                MOV     BX, HOURS       ；打印时间的小时数
                CALL    BINIDEC         ；调二进制转换为 10 进制并打印输出子程序
                MOV     DL, ':'         ；打印输出冒号 ':'
                MOV     AH, 05H
```

```
                INT     21H
                MOV     BX, MIN         ; 打印时间的分钟数
                CALL    BINIDEC
                MOV     DL, ':'
                MOV     AH, 05H
                INT     21H
                MOV     BX, SEC         ; 打印时间的秒钟数
                CALL    BINIDEC
                RET
  PRINT         ENDP                    ; PRINT 子程序结束
; ---------------------------------------------------------------------
  BINIDEC       PROC    NEAR            ; 二进制转换为 10 进制子程序
                MOV     CX, 10000D
                CALL    DEC     _DIV        ; 调除法并打印输出子程序
                MOV     CX, 1000D
                CALL    DEC     _DIV
                MOV     CX, 100D
                CALL    DEC     _DIV
                MOV     CX, 10D
                CALL    DEC     _DIV
                MOV     CX, 1D
                CALL    DEC     _DIV
                RET
  BINIDEC       ENDP                    ; BINIDEC 子程序结束
; ---------------------------------------------------------------------
  DEC_DIV       PROC    NEAR            ; 除法并打印输出子程序
                MOV     AX, BX
                MOV     DX, 0
                DIV     CX
                MOV     BX, DX          ; 余数保存在(BX)中作下一次的除法
                MOV     DL, AL          ; 商(在 00H~09H 范围内)送(DL)
                ADD     DL, 30H         ; 转换为 0~9 的 ASCII 码
                MOV     AH, 05H         ; 打印输出
                INT     21H
                RET
  DEC_DIV       ENDP                    ; DEC_DIV 子程序结束
; ---------------------------------------------------------------------
CSEG            ENDS                    ; 以上定义代码段
; ****************************************
                END     START           ; 汇编语言源程序结束
```

第九章. 习 题

9.1 INT 21H 的键盘输入功能 1 和功能 8 有什么区别？

答：键盘输入功能 1：输入字符并回显(回送显示器显示) (检测 Ctrl_Break)；
键盘输入功能 8：输入字符但不回显(也检测 Ctrl_Break)。

9.2 编写一个程序，接受从键盘输入的 10 个十进制数字，输入回车符则停止输入，然后将这些数字加密后(用 XLAT 指令变换)存入内存缓冲区 BUFFER。加密表为：

输入数字：0，1，2，3，4，5，6，7，8，9
密码数字：7，5，9，1，3，6，8，0，2，4

答：程序段如下：

```
SCODE   DB   7, 5, 9, 1, 3, 6, 8, 0, 2, 4      ; 密码数字
BUFFER  DB   10 DUP (?)
;          ┇
          MOV     SI, 0
```

```
        MOV     CX, 10
        LEA     BX, SCODE
INPUT:  MOV     AH, 1               ; 从键盘输入一个字符的功能调用
        INT     21H
        CMP     AL, 0DH             ; 输入回车符则停止输入
        JZ      EXIT
        SUB     AL, 30H             ; 是 0~9 吗?
        JB      INPUT
        CMP     AL, 09H
        JA      INPUT
        XLAT                        ; 换为密码
        MOV     BUFFER[SI], AL      ; 保存密码
        INC     SI
        LOOP    INPUT
EXIT:   RET
```

9.3 对应黑白显示器屏幕上 40 列最下边一个象素的存储单元地址是什么？

答：对应黑白显示器屏幕上 40 列最下边一个象素的存储单元地址是：B000:0F78H

9.4 写出把光标置在第 12 行，第 8 列的指令。

答：指令如下：

```
        MOV     DH, 0BH             ; 0BH=12-1
        MOV     DL, 07H             ; 07H=8-1
        MOV     BH, 0
        MOV     AH, 2               ; 置光标功能调用
        INT     10H
```

9.5 编写指令把 12 行 0 列到 22 行 79 列的屏幕清除。

答：指令如下：

```
        MOV     AL, 0               ; 清除屏幕
        MOV     BH, 07
        MOV     CH, 12              ; 左上角行号
        MOV     CL, 0               ; 左上角列号
        MOV     DH, 22              ; 右下角行号
        MOV     DL, 79              ; 右下角列号
        MOV     AH, 6               ; 屏幕上滚功能调用
        INT     10H
```

9.6 编写指令使其完成下列要求。

(1) 读当前光标位置

(2) 把光标移至屏底一行的开始

(3) 在屏幕的左上角以正常属性显示一个字母 M

答：指令序列如下：

```
(1)     MOV     AH, 3               ; 读当前光标位置，返回 DH/DL=光标所在的行/列
        MOV     BH, 0
        INT     10H
(2)     MOV     DH, 24              ; 设置光标位置
        MOV     DL, 0
        MOV     BH, 0
        MOV     AH, 2
        INT     10H
(3)     MOV     AH, 2               ; 设置光标位置
        MOV     DX, 0
        MOV     BH, 0
        INT     10H
        MOV     AH, 9               ; 在当前光标位置显示一个字符
        MOV     AL, 'M'
        MOV     BH, 0
```

```
        MOV     BL, 7
        MOV     CX, 1
        INT     10H
```

9.7 写一段程序，显示如下格式的信息：

Try again, you have n starfighters left.

其中 n 为 CX 寄存器中的 1~9 之间的二进制数。

答：程序段如下：

```
MESSAGE   DB   'Try again, you have '
CONT      DB   n
          DB   ' starfighters left.$'
;          ┇
          ADD     CL, 30H
          MOV     CONT, CL          ; 保存 ASCII 码
          LEA     DX, MESSAGE
          MOV     AH, 9             ; 显示一个字符串的 DOS 调用
          INT     21H
```

9.8 从键盘上输入一行字符，如果这行字符比前一次输入的一行字符长度长，则保存该行字符，然后继续输入另一行字符；如果它比前一次输入的行短，则不保存这行字符。按下'$'输入结束，最后将最长的一行字符显示出来。

答：程序段如下：

```
STRING  DB   0                      ; 存放字符的个数
        DB   80 DUP (?), 0DH,0AH,'$'  ; 存放前一次输入的字符串，兼作显示缓冲区
BUFFER  DB   80                     ; 输入字符串的缓冲区，最多输入 80 个字符
        DB   ?
        DB   80 DUP (20H)
;        ┇
INPUT:  LEA     DX, BUFFER          ; 输入字符串
        MOV     AH, 0AH             ; 输入字符串的 DOS 调用
        INT     21H
        LEA     SI, BUFFER+1        ; 比较字符串长度
        LES     DI, STRING
        MOV     AL, [SI]
        CMP     AL, [DI]
        JBE     NEXT
        MOV     CX, 80+1            ; 大于前次输入的字符串，更换前次的字符串
        CLD
        REP     MOVSB
NEXT:   MOV     AH, 1               ; 输入结束符吗？
        INT     21H
        CMP     AL, '$'               ; 是结束符吗？
        JNE     INPUT               ; 不是则继续输入
        LEA     DX, STRING+1        ; 显示字符串
        MOV     AH, 9               ; 显示一个字符串的 DOS 调用
        INT     21H
```

9.9 编写程序，让屏幕上显示出信息"What is the date (mm/dd/yy)?"并响铃(响铃符为 07)，然后从键盘接收数据，并按要求的格式保存在 date 存储区中。

答：程序段如下：

```
MESSAGE   DB   'What is the date (mm/dd/yy)?', 07H, '$'
DATAFLD   DB   10, 0
DATE      DB   10 DUP (' ')
;          ┇
          MOV     AH, 9             ; 显示一个字符串的 DOS 调用
          LEA     DX, MESSAGE       ; 显示字符串
          INT     21H
```

```
        MOV     AH, 0AH             ; 输入字符串的 DOS 调用
        LEA     DX, DATAFLD
        INT     21H
```

9.10 用户从键盘输入一文件并在屏幕上回显出来。每输入一行(≤80 字符)，用户检查一遍，如果用户认为无需修改，则键入回车键，此时这行字符存入 BUFFER 缓冲区保存，同时打印机把这行字符打印出来并回车换行。

答：程序段如下：

```
INAREA  DB   80                     ; 输入字符串的缓冲区，最多输入 80 个字符
ACTLEN  DB   ?
BUFFER  DB   80 DUP (?)
;            ┇
INPUT:  LEA     DX, INAREA          ; 输入字符串
        MOV     AH, 0AH             ; 输入字符串的 DOS 调用
        INT     21H
        CMP         ACTLEN, 0
        JE      EXIT
        MOV     BX, 0
        MOV     CH, 0
        MOV     CL, ACTLEN
PRINT:  MOV     AH, 5               ; 打印输出
        MOV     DL, BUFFER[BX]
        INT     21H
        INC     BX
        LOOP    PRINT
        MOV     AH, 5               ; 打印输出回车换行
        MOV     DL, 0AH
        INT     21H
        MOV     DL, 0DH
        INT     21H
        JMP     INPUT
EXIT:   RET
```

9.11 使用 MODE 命令，设置 COM2 端口的通信数据格式为：每字 8 位，无校验，1 位终止位，波特率为 1200b/s。

答：命令格式如下：

```
MODE    COM2：12，N，8，1
```

第十章. 习 题

10.1 写出指令，选择显示方式 10H，并将背景设为绿色。

答：

```
        MOV     AH, 00H
        MOV     AL, 10H             ; 选择显示方式 10H(16 色图形)
        INT     10H
        MOV     AH, 10H
        MOV     AL, 00H
        MOV     BH, 10H         ; 背景设为绿色(02H 也可以，是用 DEBUG 调试出来的)
        MOV     BL, 0               ; 选择 0 号调色板
        INT     10H
```

设置背景色也可用：

```
        MOV     AH, 0BH             ; 设置背景色和调色板
        MOV     BH, 0               ; 设置背景色功能
        MOV     BL, 8               ; 绿色背景
```

```
        INT     10H
```

10.2 如何使用 INT 10H 的功能调用改变显示方式？

答：在 AH 中设置功能号 00H，在 AL 中设置显示方式值，调用 INT 10H 即可。

10.3 VGA 独有的一种显示方式是什么？

答：像素值为 640×480，可同时显示 16 种颜色，这种显示方式(12H)是 VGA 独有的。

10.4 对于 EGA 和 VGA 显示适配器，使用显示方式 13H 时(只有 VGA 有)，显示数据存在哪里？

答：显示数据存在显示存储器里。

10.5 对于 VGA 的显示方式 13H 时存放一屏信息需要多少字节的显存？

答：需要 64000 个字节。

10.6 利用 BIOS 功能编写图形程序：设置图形方式 10H，选择背景色为蓝色，然后每行(水平方向)显示一种颜色，每 4 行重复一次，一直到整个屏幕都显示出彩条。

答：程序如下：

```
TITLE   GRAPHIX.COM
codeseg segment
        assume  cs:codeseg, ds:codeseg, ss:codeseg
        org  100h
main    proc far
        movah, 00h
        moval, 10h              ; 选择显示方式 10h(16 色图形)
        int  10h
        movah, 0bh
        movbh, 00h
        movbl, 01h              ; 背景设为蓝色
        int  10h
        movah, 0bh
        movbh, 01h
        movbl, 00h              ; 设置调色板 0#
        int  10h
        movbx, 0                  ; 显存的第 0 页
        movcx, 0                ; 起始列号为 0 列
        movdx, 0                  ; 起始行号为 0 行
line:   mov ah, 0ch             ; 写像素点
        moval, bl
        int  10h
        inc  cx
        cmpcx, 640
        jne  line
        movcx, 0                ; 起始列号为 0 列
        inc  bl
        and  bl, 03h            ; 只显示四种颜色(因此保留最低两位)
        inc  dx
        cmpdx, 350
        jne  line
        int  20h
main    endp
codeseg ends
        end  main
```

10.7 修改 10.6 题的程序，使整个屏幕都显示出纵向的彩条。

答：程序如下：

```
TITLE   GRAPHIX.COM
codeseg segment
        assume  cs:codeseg, ds:codeseg, ss:codeseg
        org  100h
```

```
main    proc far
        mov ah, 00h
        mov al, 10h                 ; 选择显示方式 10h(16 色图形)
        int  10h
        mov ah, 0bh
        mov bh, 00h
        mov bl, 01h                 ; 背景设为蓝色
        int  10h
        mov ah, 0bh
        mov bh, 01h
        mov bl, 00h                 ; 设置调色板 0#
        int  10h
        mov bx, 0                     ; 显存的第 0 页
        mov cx, 0                   ; 起始列号为 0 列
        mov dx, 0                     ; 起始行号为 0 行
line:   mov ah, 0ch                 ; 写像素点
        mov al, bl
        int  10h
        inc  dx
        cmp dx, 350
        jne  line
        mov dx, 0                     ; 起始行号为 0 行
        inc  bl
        and  bl, 03h                ; 只显示四种颜色(因此保留最低两位)
        inc  cx
        cmp cx, 640
        jne  line
        int  20h
main    endp
codeseg ends
        end  main
```

10.8 按动键盘上的光标控制键，在屏幕上下左右任一方向上绘图，每画一点之前，由数字键 0~3 指定该点的颜色值，按动 ESC 键，绘图结束，返回 DOS。

答：程序如下：

```
; DRAW—Program to draw on screen with sursor arrows
; For 640*350 color mode
up      equ  48h                    ; 向上键的扫描值
down    equ  50h                    ; 向下键的扫描值
left    equ  4bh                    ; 向左键的扫描值
right   equ  4dh                    ; 向右键的扫描值
escape  equ  1bh                    ; “Esc” character
codeseg segment
main    proc far
        assume  cs:codeseg
; clear screen by scrolling it, using ROM call
start:  mov ah, 06h
        mov al, 00h
        mov cx, 00h
        mov dl, 79
        mov dh, 24
        int  10h
; screen pointer will be in CX，DX registers；row number (0 to 350d) in DX
; coumn number (0 to 640d) in CX
        mov ah, 00h
        mov al, 10h                 ; 选择显示方式 10h(16 色图形)
        int  10h
```

```
                mov ah, 0bh
                mov bh, 00h
                mov bl, 01h                    ；背景设为蓝色
                int  10h
                mov ah, 0bh
                mov bh, 01h
                mov bl, 00h                    ；设置调色板 0#
                int  10h
                mov dx, 175                    ；设在屏幕中心
                mov cx, 320
        ；get character from keyboard
        get_char: mov ah, 0                    ；键盘输入
                int  16h
                cmp al, escape
                jz   exit
                cmp al, 33h                    ；>‘3’吗？
                jg   plot
                cmp al,30h                     ；<‘0’吗？
                jl   plot
                mov bl, al                     ；是‘0’~‘3’，设置颜色
                and  bl, 03
                jmp  get_char
        ；figure out which way to go, and draw new line
        plot:       mov al, ah
                cmp al, up
                jnz  not_up
                dec  dx
        not_up:     cmp al, down
                jnz  not_down
                inc  dx
        not_down:cmp al, right
                jnz  not_right
                inc  cx
        not_ right: cmp al, left
                jnz  write
                dec  cx
        ；use ROM routine to write dot，reguires row# in DX，col in CX，color in AL
        write:          mov al, bl
                mov ah, 0ch
                int  10h
                jmp  get_char
        exit:       int  20h
        main        endp
        codeseg     ends
                end  start
```

10.9 位屏蔽寄存器的作用是什么？在 16 色，640×480 显示方式中如何使用位屏蔽寄存器？

答：位屏蔽寄存器的作用是决定了新的像素值产生的方法。当位屏蔽寄存器的某位设为 0 时，相对应的像素值直接由锁存器写入显存；位屏蔽寄存器的某位为 1 时，所对应的像素值由锁存器中的像素值与 CPU 数据或置位/重置寄存器中相应位合并之后产生。

10.10读映像选择寄存器的作用是什么？如果 4 个位面的内容都需要读取，读映像选择寄存器应如何设置？

答：读映像选择寄存器的作用是用于选择哪一个位面的字节读入 CPU。读映像选择寄存器的 0 和 1 位，用来指定哪个位面的锁存器内容读到 CPU。如果 4 个位面的内容都需要读取，则必须对同一地址执行 4 次读操作，在每次读之前，用指令分别设置读映像选择寄存器。

10.11编写程序使一只“鸟”飞过屏幕。飞鸟的动作可由小写字母 v (ASCII 码 76H)变为破折号(ASCII

码 0C4H)来模仿，这个字符先后交替在两列显示。鸟的开始位置是 0 列 20 行，每个字符显示 0.5 秒，然后消失。

答：程序段如下：

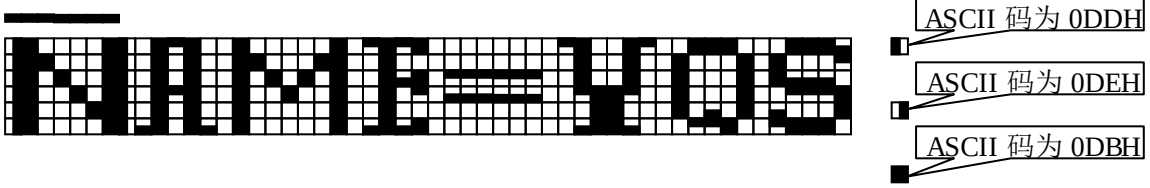
```
TITLE     Flier.EXE                    ; 飞鸟程序
; ****************************************
DSEG      SEGMENT                      ; 定义数据段
  BIRD    DB   76H, 07                 ; 小写字母 v 及属性
          DB   0C4H, 07                ; 破折号及属性
DSEG      ENDS                         ; 以上定义数据段
; ****************************************
CSEG      SEGMENT                      ; 定义代码段
  MAIN    PROC   FAR
          ASSUME   CS: CSEG, DS: DSEG
  START:  PUSH   DS                    ; 设置返回 DOS
          SUB      AX, AX
          PUSH   AX
          MOV    AX, DSEG
          MOV    DS, AX                ; 给 DS 赋值

          MOV    AH, 0FH               ; 取当前显示方式
          INT    10H
          PUSH   AX                    ; 保存当前显示方式(AL)
          MOV    AH, 0                 ; 设置彩色 80×25 文本方式
          MOV    AL, 3
          INT    10H
          MOV    DH, 20                ; 20 行
          MOV    DL, 0                 ; 0 列
BEGIN:    MOV    SI, 2                 ; 字符 v 和破折号“-”交替显示
          MOV    CX, 1                 ; 一次显示一个字符及属性
          LEA      DI, BIRD
DISP:     CMP    DL, 79                ; 飞到 79 列就退出
          JAE    EXIT
          MOV    AH, 2                 ; 置光标位置
          INT    10H
          MOV    AH, 9                 ; 在光标位置显示字符及属性
          MOV    AL, [DI]              ; 取显示字符及属性
          MOV    BL, [DI+1]
          INT    10H
          CALL   DELAY                 ; 延时 0.5 秒
          MOV    AH, 9                 ; 在光标位置显示字符及属性
          MOV    AL, ‘ ’               ; 显示空格，擦除该位置的字符
          MOV    BL, 7
          INT    10H
          INC    DL                    ; 飞到下一列
          ADD    DI, 2
          DEC      SI
          JNZ    DISP
          JMP    BEGIN
  EXIT:   POP    AX                    ; 恢复当前显示方式(AL)
          MOV    AH, 0
          INT    10H
          RET                          ; 返回 DOS
  MAIN    ENDP
; ---------------------------------------------------------------------
  DELAY   PROC   NEAR                  ; 延时 0.5s 子程序
```

```
                PUSH    CX
                PUSH    DX
                MOV     DX, 50          ；延时 0.5s
DEL1:           MOV     CX, 2801        ；延时 10ms
DEL2:           LOOP    DEL2
                DEC       DX
                JNZ     DEL1
                POP     DX
                POP     CX
                RET
DELAY           ENDP                    ；DELAY 子程序结束
；-------------------------------------------------------------------------
CSEG            ENDS                    ；以上定义代码段
；*****************************************
                END     START           ；汇编语言源程序结束
```

10.12用图形文本的方法设计“Name=XXX”(X 为你自己姓名的缩写)，并将其数据编码定义在一个数组中。

答：用图形文本的方法设计“NAME=YQS”的程序和数组如下：

显示格式如下：




```
TITLE           NAME_YQS.EXE            ；显示“NAME=YQS”的程序
；*****************************************
；Graphics block message for the words shooting NAME=YQS
；00H→end of massage，0FFH→end of screen line
DSEG            SEGMENT                 ；定义数据段
 NAME_YQS       DB   2                  ；Start row（开始行）
                DB   2                  ；Start column（开始列）
                DB   1000 0011B         ；Color attribute
                DB   ‘Shooting’,0FFH,0FFH；显示“Shooting”
                DB   7 DUP(0DCH),0FFH,0FFH
；Graphics encoding of the word NAME=YQS  using IBM character set
                DB   0DEH, 0DBH, 4 DUP(20H), 0DBH, 0DDH, 20H, 0DBH, 0DFH, 0DBH
                DB   20H, 20H, 0DBH, 5 DUP(20H), 0DBH, 20H, 2 DUP(0DFH, 0DBH)
                DB   8 DUP(20H), 0DFH, 0DBH, 20H, 20H, 0DBH, 0DFH, 20H, 20H
                DB   3 DUP(0DBH), 3 DUP(20H), 3 DUP(0DBH), 0DCH, 0FFH

                DB   0DEH, 0DBH, 0DBH, 3 DUP(20H), 0DBH, 0DDH, 2 DUP(20H, 0DBH)
                DB   20H, 20H, 0DBH, 0DBH, 3 DUP(20H), 0DBH, 0DBH, 20H, 20H, 0DBH
                DB   11 DUP(20H), 3 DUP(0DBH, 20H, 20H), 20H, 0DBH, 20H, 0DBH
                DB   3 DUP(20H), 0DFH, 0FFH

                DB   0DEH, 0DBH, 20H, 0DBH, 20H, 20H, 0DBH, 0DDH, 2 DUP(20H, 0DBH)
                DB   20H, 4 DUP(20H, 0DBH), 20H, 20H, 0DBH, 0DCH, 0DBH, 20H
                DB   7 DUP(0DFH), 3 DUP(20H, 20H, 0DBH), 3 DUP(20H), 0DBH, 20H, 0DBH
                DB   3 DUP(0DCH), 20H, 0FFH

                DB   0DEH, 0DBH, 20H, 20H, 0DBH, 20H, 0DBH, 0DDH, 20H, 0DBH, 0DFH
```

```
                DB   0DBH, 4 DUP(20H, 20H, 0DBH), 20H, 0DFH, 20H, 7 DUP(0DCH), 20H
                DB   20H, 0DFH, 0DBH, 0DBH, 0DFH, 20H, 20H, 0DBH, 3 DUP(20H), 0DBH
                DB   20H, 20H, 3 DUP(0DFH), 0DBH, 0FFH

                DB   0DEH, 0DBH, 3 DUP(20H), 0DBH, 0DBH, 0DDH, 2 DUP(20H, 0DBH), 20H
                DB   20H, 0DBH, 5 DUP(20H), 2 DUP(0DBH, 20H, 20H), 10 DUP(20H), 0DBH
                DB   0DBH, 3 DUP(20H), 0DBH, 20H, 0DCH, 20H, 0DBH, 20H, 0DCH
                DB   3 DUP(20H), 0DBH, 0FFH

                DB   0DEH, 0DBH, 4 DUP(20H), 0DBH, 0DDH, 0DCH, 0DBH, 20H, 0DBH
                DB   0DCH, 20H, 0DBH, 5 DUP(20H), 0DBH, 20H, 2 DUP(0DCH, 0DBH)
                DB   9 DUP(20H), 0DCH, 0DBH, 0DBH, 0DCH, 3 DUP(20H), 0DFH, 0DFH
                DB   0DBH, 20H, 20H, 0DFH, 3 DUP(0DBH), 20H, 0FFH
                DB   00                    ; 结束显示标志
 START_COL DB   ?
DSEG            ENDS                       ; 以上定义数据段
; ****************************************
; Text display procedures: display a message on the graphics screen
CSEG            SEGMENT                    ; 定义代码段
  MAIN          PROC    FAR
                ASSUME  CS: CSEG, DS: DSEG
  START:        PUSH    DS                 ; 设置返回 DOS
                SUB             AX, AX
                PUSH    AX
                MOV     AX, DSEG
                MOV     DS, AX

                LEA             DI, NAME_YQS
                MOV     DH, [DI]           ; Get row into DH
                INC     DI                 ; Bump pointer
                MOV     DL, [DI]           ; And column into DL
                MOV     START_COL, DL      ; Store start column
                MOV     AH, 2              ; Set cursor position
                MOV     BH, 0              ; Page 0
                INT     10H
                INC     DI                 ; Bump pointer to attribute
                MOV     BL, [DI]           ; Get color code into BL
  Char_write:   INC     DI                 ; Bump to message start
                MOV     AL, [DI]           ; Get character
                CMP     AL, 0FFH           ; End of line?
                JE      BUMP_ROW           ; Next row
                CMP     AL, 0              ; Test for terminator
                JE      END_TEXT           ; Exit routine
                CALL    SHOW_CHAR
                JMP     CHAR_WRITE
  END_TEXT:     RET                        ; 返回 DOS
  Bump_row:     INC DH                     ; Row control register
                MOV     DL, START_COL      ; Column control to start column
                MOV     AH, 2              ; Set cursor position
                MOV     BH, 0              ; Page 0
                INT     10H
                JMP     CHAR_WRITE
  MAIN          ENDP
; ---------------------------------------------------------------------------
; Display character in AL and using the color code in BL
  Show_char     PROC    NEAR               ; 显示字符子程序
```

```
                MOV     AH, 9           ; BIOS service request number
                MOV     BH, 0           ; Page 0
                MOV     CX, 1           ; No repeat
                INT     10H
        ; Bump cursor
                INC     DL
                MOV     AH, 2           ; Set cursor position
                MOV     BH, 0           ; Page 0
                INT     10H
                RET
          Show_char ENDP                ; SHOW_CHAR 子程序结束
        ; ------------------------------------------------------------------------
        CSEG    ENDS                    ; 以上定义代码段
        ; ****************************************
                END     START           ; 汇编语言源程序结束
```

10.13游戏程序常常用随机数来控制其图形在屏幕上移动。请编写一程序，用随机数来控制笑脸符(ASCII 码 02H)显示的位置。笑脸符每次显示的列号总是递增 1。而行的位置可能是前次的上一行，下一行或同一行，这根据随机数是 0、1 或 2 来决定，当行号变为 0、24 或列号变为 79 时显示结束。笑脸在每个位置上显示 0.25s。(提示：INT 1AH 的 AH=0 是读当前时间的功能调用，利用该功能返回的随时都在变化的时间值作为产生随机数的基数。)

答：程序段如下：

```
        TITLE   Disp_Laugh.EXE          ; 笑脸显示程序
        ; ****************************************
        CSEG    SEGMENT                 ; 定义代码段
          MAIN  PROC    FAR
                ASSUME  CS: CSEG
          START: PUSH   DS              ; 设置返回 DOS
                SUB     AX, AX
                PUSH    AX

                MOV     AH, 0FH         ; 取当前显示方式
                INT     10H
                PUSH    AX              ; 保存当前显示方式(AL)
                MOV     AH, 0           ; 设置彩色 80×25 文本方式
                MOV     AL, 3
                INT     10H

                MOV     CX, 1           ; 一次显示一个笑脸字符及属性
                MOV     DH, 12H         ; 12 行，从屏幕左边的中间开始
                MOV     DL, 0           ; 0 列
        BEGIN:  CMP     DL, 79          ; 移到 79 列就退出
                JAE     EXIT
                CMP     DH, 0           ; 移到第 0 行就退出
                JBE     EXIT
                CMP     DH, 24          ; 移到第 24 行就退出
                JAE     EXIT
                MOV     AH, 2           ; 置光标位置
                INT     10H
                MOV     AH, 9           ; 在光标位置显示字符及属性
                MOV     AL, 02H         ; 取笑字符及属性
                MOV     BL, 7
                INT     10H
                CALL    DELAY           ; 延时 0.25 秒
                MOV     AH, 9           ; 在光标位置显示字符及属性
```

```
                MOV     AL, ' '         ; 显示空格，擦除该位置的字符
                MOV     BL, 7
                INT     10H
                INC     DL              ; 移到下一列
                PUSH    DX
                MOV     AH, 0           ; 读当前时间, CH:CL=时:分，DH:DL=秒:1/100 秒
                                        ; 产生随机数基数
                INT     1AH
                MOV     AX, DX
                POP     DX
                AND     AL, 03H         ; 随机数为 1/100 秒的最低两位
                JZ      DOWN            ; 随机数的最低两位为 0 则下降一行
                CMP     AL, 1
                JNZ     LEVEL           ; 随机数的最低两位为≥2 则水平移动
                DEC     DH                ; 随机数的最低位为 1 则上跳一行
                JMP     BEGIN
DOWN:           INC     DH
LEVEL:      JMP     BEGIN
EXIT:           POP     AX              ; 恢复当前显示方式(AL)
                MOV     AH, 0
                INT     10H
                RET                     ; 返回 DOS
MAIN            ENDP
; ---------------------------------------------------------------------------
DELAY           PROC    NEAR            ; 延时 0.25s 子程序
                PUSH    CX
                PUSH    DX
                MOV     DX, 25          ; 延时 0.25s
DEL1:           MOV     CX, 2801        ; 延时 10ms
DEL2:           LOOP    DEL2
                DEC     DX
                JNZ     DEL1
                POP     DX
                POP     CX
                RET
DELAY           ENDP                    ; DELAY 子程序结束
; ---------------------------------------------------------------------------
CSEG            ENDS                    ; 以上定义代码段
; ****************************************
                END     START           ; 汇编语言源程序结束
```

10.14分配给 PC 机主板上的 8253/54 定时器的端口地址是什么？

答：8253/54 定时器的 3 个独立计数器 Counter0、Counter1 和 Counter2 的端口地址分别为 40H、41H 和 42H。8253/54 内部还有一个公用的控制寄存器，端口地址为 43H。

10.158253/54 定时器的三个计数器，哪一个用于扬声器？它的端口地址是什么？

答：8253/54 定时器的计数器 Counter2 用于扬声器，它的端口地址为 42H。

10.16下面的代码是利用监控端口 61H 的 PB4 来产生延迟时间的，它适用于所有的 286、386、Pentium PC 及兼容机。请指出该程序的延迟时间是多少？

```
        MOV     DL, 200
BACK:   MOV     CX, 16572
WAIT:   IN      AL, 61H
        AND     AL, 10H
        CMP     AL, AH
        JE      WAIT
        MOV     AH, AL
```

```
        LOOP    WAIT
        DEC     DL
        JNZ     BACK
```

答：该程序的延迟时间是 200×16572×15.08μs＝49981152μs≈50s。

10.17在 PC 机上编写乐曲程序“Happy Birthday”，乐曲的音符及音频如下：

歌词	音符	音频	节拍	歌词	音符	音频	节拍	歌词	音符	音频	节拍
hap	C	262	1/2	day	C	262	1	so	D	294	3
py	C	262	1/2	to	G	392	1	hap	Bb	466	1/2
birth	D	294	1	you	F	349	2	py	Bb	466	1/2
day	C	262	1	hap	C	262	1/2	birth	A	440	1
to	F	349	1	py	C	262	1/2	day	C	262	1
you	E	330	2	birth	D	294	1	to	G	392	1
hap	C	262	1/2	day	A	440	1	you	F	349	2
py	C	262	1/2	dear	F	349	1				
birth	D	294	1	so	E	330	1				

答：程序如下：

```
TITLE       MUSIC  —  A music of ‘Happy Birthday’   ；连接时需加上 GENSOUND 程序
EXTRN       SOUNDF: FAR                  ；SOUNDF 是外部过程——通用发声程序
；*****************************************
STACK       SEGMENT   PARA   STACK ‘STACK’   ；定义堆栈段
            DB   64 DUP (‘STACK…’)
STACK       ENDS                         ；以上定义堆栈段
；*****************************************
DSEG        SEGMENT   PARA   ‘DATA’      ；定义数据段
  MUS_FREQ DW   262, 262, 294, 262, 349, 330, 262, 262, 294, 262,392, 349, 262, 262
            DW   294, 440, 349, 330, 294, 466, 466,440,262, 392, 349, -1
  MUS_TIME DW   25, 25, 50, 50, 50, 100
            DW   25, 25, 50, 50, 50, 100
            DW   25, 25, 50, 50, 50, 50, 150
            DW   25, 25, 50, 50, 50, 100
DSEG        ENDS                         ；以上定义数据段
；*****************************************
CSEG        SEGMENT   PARA ‘CODE’        ；定义代码段
            ASSUME   CS: CSEG, DS: DSEG, SS: STACK
  MUSIC     PROC   FAR
            PUSH   DS                    ；设置返回 DOS
            SUB        AX, AX
            PUSH   AX
            MOV    AX, DSEG
            MOV    DS, AX                ；给 DS 赋值

            LEA        SI, MUS_FREQ        ；取发声的频率(音阶)表首地址
            LEA        BP, MUS_TIME        ；取发声的节拍(时间)表首地址
FREQ:       MOV    DI, [SI]              ；读取频率值
            CMP    DI, -1                ；歌曲结束了吗？
            JE     END_MUS
            MOV    BX, DS:[BP]           ；读取节拍
            CALL   SOUNDF                  ；调通用发声子程序
            ADD    SI, 2
            ADD    BP, 2
            JMP    FREQ
  END_MUS:  RET                           ；返回 DOS
  MUSIC     ENDP
CSEG        ENDS                         ；以上定义代码段
；*****************************************
```

```
                END     MUSIC                   ；汇编语言源程序结束
```

以下是 SOUNDF ——外部的通用发声子程序（教材 392 页）

```
TITLE       SOUNDF ―― 通用发声子程序
; ****************************************
PUBLIC      SOUNDF                              ；定义为公共过程
; ****************************************
CSEG1       SEGMENT  PARA 'CODE'    ；定义代码段
            ASSUME   CS: CSEG1
  SOUNDF    PROC     FAR
            PUSH     AX
            PUSH     BX
            PUSH     CX
            PUSH     DX
            PUSH     DI
            MOV      AL, 0B6H               ；写定时器 8253 的工作方式
            OUT      43H, AL
            MOV      DX, 12H                ；根据频率求 8253 的计数值，即 533H*896/freq
            MOV      AX, 533H*896           ；(DX),(AX)=123280H=533H*896
            DIV      DI                     ；(DI) = freq
            OUT      42H, AL                ；向 8253 送计数值
            MOV      AL, AH
            OUT      42H, AL
            IN       AL, 61H                ；取 8255 的 PB 口当前内容，并保护
            MOV      AH, AL
            OR       AL, 3                  ；开始发声，PB1=1，PB0=1
            OUT      61H, AL
WAIT1:      MOV      CX, 663                ；延时(BX)×10ms
            CALL     WAITF
            MOV      AL, AH
            AND      AL, 0FCH               ；停止发声，PB1=0，PB0=0
            OUT      61H, AL
            POP      DI
            POP      DX
            POP      CX
            POP      BX
            POP      AX
            RET
  SOUNDF    ENDP
; ****************************************
  WAITF     PROC     NEAR
            PUSH     AX
  WAITF1:   IN       AL, 61H
            AND      AL, 10H
            CMP      AL, AH
            JE       WAITF1
            MOV      AH, AL
            LOOP     WAITF1
            POP      AX
            RET
  WAITF     ENDP
CSEG1       ENDS                            ；以上定义代码段
; ****************************************
            END
```

10.18编写用键盘选择计算机演奏歌曲的程序。首先在屏幕上显示出歌曲名单如下：

```
    A   MUSIC 1
```

B MUSIC 2
C MUSIC 3

当从键盘上输入歌曲序号 A，B 或 C 时，计算机则演奏所选择的歌曲，当在键盘上按下 0 键时，演奏结束。

答：程序段如下：

```
MUS_LST  DB  'A  MUSIC 1', 0DH, 0AH
         DB  'B  MUSIC 2', 0DH, 0AH
         DB  'C  MUSIC 3', 0DH, 0AH
         DB  '0  END', 0DH, 0AH, '$'
           ⋮
         MOV   AH, 09              ; 显示字符串的 DOS 功能调用
         LEA       DX, MUS_LIST
         INT   21H
INPUT:   MOV   AH, 1               ; 键盘输入一个字符的 DOS 功能调用
         INT   21H
         CMP   AL, '0'             ; 结束演奏吗?
         JE    EXIT
         OR    AL, 0010 0000B      ; 变为小写字母
         CMP   AL, 'a'             ; 演奏歌曲 a 吗?
         JNZ   B0
         CALL  MUSIC1              ; 去演奏歌曲 A
         JMP   INPUT
B0:        CMP   AL, 'b'           ; 演奏歌曲 b 吗?
         JNZ   C0
         CALL  MUSIC2              ; 去演奏歌曲 B
         JMP   INPUT
C0:        CMP   AL, 'c'           ; 演奏歌曲 c 吗?
         JNZ   INPUT
         CALL  MUSIC3              ; 去演奏歌曲 C
         JMP   INPUT
EXIT:    RET                       ; 返回
```

第十一章. 习 题

11.1 写出文件代号式磁盘存取操作的错误代码：

(1) 非法文件代号 (2) 路径未发现 (3) 写保护磁盘

答：错误代码为：

(1) 06 (2) 03 (4) 19

11.2 使用 3CH 功能建立一文件，而该文件已经存在，这时会发生什么情况？

答：此操作将文件长度置为 0，写新文件，原文件内容被清除。

11.3 从缓冲区写信息到一个文件，如果没有关闭文件，可能会出现什么问题？

答：文件结尾的部分信息就没有被写入磁盘，从而造成写入的文件不完整。

11.4 下面的 ASCIZ 串有什么错误？

```
PATH_NAME   DB  'C:\PROGRAMS\TEST.DAT'
```

答：此 ASCIZ 串的最后少了一个全 0 字节，应改为：

```
PATH_NAME   DB  'C:\PROGRAMS\TEST.DAT', 0
```

11.5 下面为保存文件代号定义的变量有什么错误？

```
FILE_HNDL   DB  ?
```

答：文件代号是字类型，因此应改为：

```
FILE_HNDL   DW  ?
```

11.6 在 ASCPATH 字节变量中为驱动器 D 的文件 PATIENT.LST，请定义 ASCIZ 串。

答：ASCPATH DB 'D:\PATIENT.LST', 0

11.7 对 11.6 题中的文件，它的每个记录包含：

病例号(patient number)： 5 字符， 姓名(name)： 20 字符，
城市(city)： 20 字符， 街道(street address)： 20 字符，
出生年月(mmddyy)： 6 字符， 性别(M/Fcode)： 1 字符，
病房号(room number)： 2 字符， 床号(bed number)： 2 字符，

(1) 定义病人记录的各个域 (2) 定义保存文件代号的变量 FHANDLE
(3) 建文件 (4) 把 PATNTOUT 中的记录写入
(5) 关文件 (6) 以上文件操作包括测试错误

答：(1)
```
PATNTOUT   EQU   THIS   BYTE
patient    DB    5    DUP (?)
name       DB    20   DUP (?)
city       DB    20   DUP (?)
street     DB    20   DUP (?)
mmddyy     DB    6    DUP (?)
M_Fcode    DB    ?
room       DB    2    DUP (?)
bed        DB    2    DUP (?), 0AH, 0DH
COUNT = $－PATNTOUT                ; 记录长度
```
(2)
```
FHANDLE    DW    ?
```
(3)
```
MOV    AH, 3CH                     ; 建文件功能
MOV    CX, 00                      ; 普通文件属性
LEA    DX, ASCPATH
INT    21H
JC     ERROR
MOV    FHANDLE, AX                 ; 保存文件代号
```
(4)
```
MOV    AH, 40H                     ; 写文件功能
MOV    BX, FHANDLE                 ; 取文件代号
MOV    CX, COUNT                   ; 记录长度
LEA    DX, PATNTOUT                ; 记录的首地址
INT    21H
JC     ERROR
CMP    AX, COUNT                   ; 所有的字节都写入了吗?
JNE    ERROR1
```
(5)
```
MOV    AH, 3EH                     ; 关闭文件功能
MOV    BX, FHANDLE                 ; 取文件代号
INT    21H
JC     ERROR
```
(6) 文件操作的测试错误已包括在(3)、(4)、(5)的操作中。

11.8 对 11.7 题的文件，用文件代号式编写一个完整的读文件程序，读出的每个记录存入 PATNTIN 并在屏幕上显示。

答：程序如下：
```
TITLE    READDISP.EXE                ; 利用文件代号式顺序读文件程序
; Read disk records created by hancreat
; ----------------------------------------------------
         .model   small
         .stack   100h
         .data
endcde   db   0                      ; 结束处理指示
fhandle  dw   ?
patntin  db   80 DUP(' ')            ; DTA
ascpath  db   'd:\patient.lst', 0
openmsg  db   '***open error***', 0dh, 0ah
readmsg  db   '***read error***', 0dh, 0ah
row      db   0
; ----------------------------------------------------
```

```
            .code
begin       proc  far
            mov   ax, @data
            mov   ds, ax
            mov   es, ax

            mov   ax, 0600h
            call  screen                ; 清屏
            call  curs                  ; 设置光标
            call  openh                 ; 打开文件，设置 DTA
            cmp   endcde, 0             ; 打开错误吗？
            jnz   a0                    ; 错误，转结束
contin:     call  readh                 ; 读磁盘记录
            cmp   endcde, 0             ; 读错误吗？
            jnz   a0                    ; 错误，转结束
            call  disph                 ; 没错，显示记录
            jmp   contin
a0:         mov   ax, 4c00h             ; 退出程序，返回 DOS
            int   21h
begin       endp
; ---------------------------------------------------
; 打开文件
openh       proc  near
            mov   ah, 3dh
            mov   al, 0
            lea   dx, ascpath
            int   21h
            jc    b1                    ; 打开错误吗？
            mov   fhandle, ax           ; 没有错，保存文件代号
            ret
b1:         mov   endcde, 01            ; 打开错误，指示结束处理
            lea   dx, openmsg
            call  errm                  ; 显示出错信息
            ret
openh       endp
; ---------------------------------------------------
; 读磁盘记录
readh       proc  near
            mov   ah, 3fh
            mov   bx, fhandle
            mov   cx, 80
            lea   dx, patntin
            int   21h
            jc    c1                    ; 读错误吗？
            cmp   ax, 0                 ; 文件已读完吗？
            je    c2                    ; 读完，退出
            ret
c1:         lea   dx, openmsg           ; 读错误
            call  errm                  ; 显示出错信息
c2:         mov   endcde, 01            ; 读错误或文件读完，指示结束处理
            ret
readh       endp
; ---------------------------------------------------
; 显示记录
disph           proc  near
            mov   ah, 40h               ; 向标准输出设备(文件代号=01)写文件
```

```
                mov   bx, 01                    ; 标准输出设备的文件代号=01
                mov   cx, 80
                lea   dx, patntin
                int   21h
                cmp   row, 24                   ; 已到屏幕底部吗?
                jae   d1                        ; 已到屏幕底部，退出
                inc   row
                ret
d1:             mov   ax, 0601h
                call  screen                    ; 屏幕上卷一行
                call  curs                      ; 设置光标
                ret
disph           endp
; ------------------------------------------------------
; 屏幕上卷
screen  proc    near                            ; 入口参数为 ax
                mov   bh, 1eh                   ; 设置颜色
                mov   cx, 0                     ; 屏幕左上角
                mov   dx, 184fh                 ; 屏幕右下角
                int   10h
                ret
screen  endp
; ------------------------------------------------------
; 设置光标
curs    proc    near
                mov   ah, 2                     ; 设置光标
                mov   bh, 0
                mov   dh, row                   ; 行号
                mov   dl, 0                     ; 列号
                int   10h
                ret
curs    endp
; ------------------------------------------------------
; 显示出错信息
errm    proc    near
                mov   ah, 40h                   ; 向标准输出设备(文件代号=01)写文件
                mov   bx, 01                    ; 标准输出设备的文件代号=01
                mov   cx, 20
                int   21h
                ret
errm    endp
; ------------------------------------------------------
                end   begin
```

11.9 编写建立并写入磁盘文件的程序。允许用户从键盘键入零件号(3 字符)，零(配)件名称(12 字符)，单价(1 个字)。程序使用文件代号式建立含有这些信息的文件。注意要把单价从 ASCII 码转换为二进制数。下面是输入的例子：

part#	Description	price	part#	Description	price
023	Assemblers	00315	122	Lifters	10520
024	Linkages	00430	124	Processors	21335
027	Compilers	00525	127	Labtlers	00960
049	Compressors	00920	232	Bailers	05635
114	Extractors	11250	237	Grinders	08250
117	Haulers	00630	999		000

答：程序如下：

```
TITLE   HANCREAT.EXE                ;利用文件代号式建立文件程序
```

```
;-----------------------------------------------------
        .model small
        .stack  100h
        .data
prompt1 db   'Please input Part#: $'          ;提示输入零件号
prompt2 db   'Please input Description: $'        ;提示输入零件名称
prompt3 db   'Please input Price: $'          ;提示输入单价
maxlen  db   13                               ;最大输入长度，输入字符串功能的缓冲区
actlen  db   ?                                ;实际输入长度
buffer  db   13 DUP ('    ')                  ;输入字符串缓冲区
crlf    db   0dh, 0ah, '$'
pathname db  'filename.lst', 0
handle  dw   ?
dta     db   19 DUP ('    ')                  ;DTA
errcde  db   0                                ;错误处理指示
opnmsg  db   '***open error***', 0dh, 0ah
wrtmsg  db   '***write error***', 0dh, 0ah
;-----------------------------------------------------
        .code
begin   proc   far
        mov    ax, @data
        mov    ds, ax
        mov    es, ax

        mov    ax, 0600h
        call   scren                          ;清屏
        call   curs                           ;设置光标
        call   creath                         ;建立文件
        cmp    errcde, 0                      ;建立错误吗？
        jnz    a0                             ;错误，转结束
contin: call   proch                          ;记录处理
        cmp    actlen, 0                      ;输入的字符串长度为 0，结束输入吗？
        jne    contin                         ;不结束，继续
        call   clseh                          ;结束输入，关闭文件
a0:     mov    ax, 4c00h                      ;退出程序，返回 DOS
        int    21h
begin   endp
;-----------------------------------------------------
;建立文件
creath  proc   near
        mov    ah, 3ch
        mov    cx, 0                          ;普通属性
        lea    dx, pathname
        int    21h
        jc     bbb                            ;建立文件错误吗？
        mov    handle, ax                     ;没有错，保存文件代号
        ret
bbb:    lea    dx, opnmsg                     ;建立文件错误
        call   errm                           ;显示出错信息
        ret
creath  endp
;-----------------------------------------------------
;接收输入
proch   proc   near
        cld
        lea    di, dta                        ;在 di 中设置 dta 的首地址
```

```
            lea     dx, prompt1             ;输入零件号
            mov     bx, 3                   ;零件号最多 3 个字符
            call    in_proc
            jc      exit                    ;没有输入，结束
            lea     dx, prompt2             ;输入零件名称
            mov     bx, 12                  ;零件名称最多 12 个字符
            call    in_proc
            jc      exit                    ;没有输入，结束
            lea     dx, prompt3             ;输入单价
            mov     bx, 5                   ;零件单价最多 5 个十进制字符(相当于一个二进制字)
            call    in_proc
            call    dec_bin                     ;将十进制的单价转换为二进制的单价
            mov     word ptr [dta+17], 0a0dh ;在 DTA 的最后插入回车换行符
            call    writh                   ;用文件代号法写记录
exit:       ret
proch       endp
;-------------------------------------------------------
;输入字符串子程序
in_proc     proc    near
            mov     ah, 09h                 ;显示提示信息
            int     21h
            push    di
            lea     di, buffer              ;在 buffer 中填入空格符
            mov     cl, maxlen
            mov     ch, 0
            mov     al, '  '
            rep     stosb
            pop     di
            mov     ah, 0ah                 ;输入字符串
            lea     dx, maxlen
            int     21h
            call    disp_crlf
            cmp     actlen, 0               ;实际输入字符数=0，则没有输入，结束
            je      end_in
            push    di
            lea     di, buffer              ;在 buffer 的后面填入空格符
            mov     al, actlen
            mov     ah, 0
            add     di, ax
            mov     cl, maxlen
            mov     ch, 0
            mov     al, actlen
            sub     cl, al
            mov     al, '  '
            rep     stosb
            pop     di
            lea     si, buffer              ;将 buffer 缓冲区内容送入 dta
            mov     cx, bx
            rep     movsb                   ;将输入内容送入 dta
            clc                             ;有输入字符，返回(cf)=0
            jmp     in_end
end_in:     stc                             ;没有输入字符，返回(cf)=1
in_end:     ret
in_proc     endp
;-------------------------------------------------------
;将十进制的单价转换为二进制的单价子程序
```

```
dec_bin   proc   near
          mov    bx, 0
          mov    si, 0
          mov    cx, 5
transfer: mov    al, buffer[si]          ;从十进制的高位到低位取数
          cmp    al, 0dh                 ;是回车吗?
          je     dec_bin1
          cmp    al, '  '                ;是空格吗?
          je     dec_bin1
          and    al, 0fh                 ;将 ascii 码转换为十进制数
          mov    ah, 0
          push   cx
          xchg   ax, bx                  ;十进制数高位×10+低位 = 二进制数
          mov    cx, 10
          mul    cx
          xchg   ax, bx
          add    bx, ax                  ;转换的二进制数在(bx)中
          pop    cx
          inc    si
          loop   transfer
dec_bin1: mov    word ptr [dta+15], bx   ;存入单价到 dta 中的单价位置
          ret
dec_bin   endp
;-------------------------------------------------
;用文件代号法写记录
writh     proc   near
          mov    ah, 40h
          mov    bx, handle
          mov    cx, 19
          lea    dx, dta
          int    21h
          jnc    ddd                     ;写文件错误吗?
          lea    dx, wrtmsg
          call   errm                    ;显示出错信息
          mov    actlen, 0
ddd:      ret
writh     endp
;-------------------------------------------------
;用文件代号法关闭文件
clseh     proc   near
          mov    dta, 1ah                ;写文件结束符 1ah
          call   writh
          mov    ah, 3eh
          mov    bx, handle
          int    21h
          ret
clseh     endp
;-------------------------------------------------
;屏幕上卷
scren     proc   near                        ;入口参数为 ax
          mov    bh, 1eh                 ;设置颜色
          mov    cx, 0                   ;屏幕左上角
          mov    dx, 184fh               ;屏幕右下角
          int    10h
          ret
scren     endp
;-------------------------------------------------
```

```
;设置光标
curs    proc    near
        mov     ah, 2                   ;设置光标
        mov     bh, 0
        mov     dh, 0                   ;行号
        mov     dl, 0                   ;列号
        int     10h
        ret
curs    endp
;-------------------------------------------------
;显示出错信息
errm    proc    near
        mov     ah, 40h                 ;向标准输出设备(文件代号=01)写文件
        mov     bx, 01                  ;标准输出设备的文件代号=01
        mov     cx, 20
        int     21h
        mov     errcde, 01              ;错误代码置 1
        ret
errm    endp
;-------------------------------------------------
disp_crlf proc  near                        ; 显示回车换行符子程序
        lea     dx, crlf
        mov     ah, 09h
        int     21h
        ret
disp_crlf endp                          ; disp_crlf 子程序结束
; -------------------------------------------------
        end     begin                       ;汇编语言源程序结束
```

11.10编写一个程序使用文件代号式读出并显示 11.9 题建立的文件。注意，要把二进制数表示的单价转换为 ASCII 码。

答：用文件代号式读出并显示文件，程序如下：

```
TITLE   HANDREAD.EXE                ;利用文件代号式顺序读并显示文件程序
;Read disk records created by hancreat
;-------------------------------------------------
        .model small
        .stack 100h
        .data
endcde  db      0                   ;结束处理指示
crlf    db      0dh, 0ah, '$'
pathname db     'filename.lst', 0
message db      '       Part#       Description         Price',0dh,0ah,'$'
handle  dw      ?
tackline db     '    |    $'
dta     db      19 DUP (' ')        ;DTA
errcde  db      0                   ;错误处理指示
opnmsg  db      '***open error***', 0dh, 0ah
readmsg db      '***read error***', 0dh, 0ah
row             db 0
;-------------------------------------------------
        .code
begin   proc    far
        mov     ax, @data
        mov     ds, ax
        mov     es, ax

        mov     ax, 0600h
```

```
            call    screen              ;清屏
            call    curs                ;设置光标
            lea     dx, message         ;显示标题
            mov     ah, 09h
            int     21h
            inc     row
            call    openh               ;打开文件，设置 DTA
            cmp     endcde, 0           ;打开错误吗？
            jnz     a0                  ;错误，转结束
contin:     call    readh               ;读磁盘记录
            cmp     endcde, 0           ;读错误吗？
            jnz     a0                  ;错误，转结束
            call    disph               ;没错，显示记录
            jmp     contin
a0:         mov     ax, 4c00h           ;退出程序，返回 DOS
            int     21h
begin       endp
;-------------------------------------------------------
;打开文件
openh       proc    near
            mov     ah, 3dh
            mov     al, 0
            lea     dx, pathname
            int     21h
            jc      bbb                 ;打开错误吗？
            mov     handle, ax          ;没有错，保存文件代号
            ret
bbb:        mov     endcde, 01          ;打开错误，指示结束处理
            lea     dx, readmsg
            call    errm                ;显示出错信息
            ret
openh       endp
;-------------------------------------------------------
;读磁盘记录
readh       proc    near
            mov     ah, 3fh
            mov     bx, handle
            mov     cx, 19
            lea     dx, dta
            int     21h
            jc      c1                  ;读错误吗？
            cmp     ax, 0               ;文件已读完吗？
            je      c2                  ;读完，退出
            cmp     dta, 1ah                ;文件结束符吗?
            Je      c2
            ret
c1:         lea     dx, opnmsg          ;读错误
            call    errm                ;显示出错信息
c2:         mov     endcde, 01          ;读错误或文件读完，指示结束处理
            ret
readh       endp
;-------------------------------------------------------
;显示记录
disph       proc    near
            lea     dx, tackline        ;显示输出“   |   ”
            mov     ah, 09h
```

```
            int     21h
            mov     ah, 40h                 ;向标准输出设备(文件代号=01)写文件
            mov     bx, 01                  ;标准输出设备的文件代号=01
            mov     cx, 3
            lea     dx, dta
            int     21h
            lea     dx, tackline            ;显示输出“    |    ”
            mov     ah, 09h
            int     21h
            mov     ah, 40h                 ;向标准输出设备(文件代号=01)写文件
            mov     bx, 01                  ;标准输出设备的文件代号=01
            mov     cx, 12
            lea     dx, dta+3
            int     21h
            lea     dx, tackline            ;显示输出“    |    ”
            mov     ah, 09h
            int     21h
            mov     si, word ptr [dta+15]
            call    bin_dec                     ;转换为十进制数显示
            lea     dx, tackline            ;显示输出“    |    ”
            mov     ah, 09h
            int     21h
            call    disp_crlf
            cmp     row, 24                 ;已到屏幕底部吗?
            jae     ddd                     ;已到屏幕底部，退出
            inc     row
            ret
ddd:        mov     ax, 0601h
            call    screen                  ;屏幕上卷一行
            call    curs                    ;设置光标
            ret
disph       endp
;-------------------------------------------------------
;将二进制的单价转换为十进制的单价并显示子程序
bin_dec     proc    near
            push    cx
            mov     cx, 10000d
            call    dec_div                     ;调除法并显示输出子程序
            mov     cx, 1000d
            call    dec_div
            mov     cx, 100d
            call    dec_div
            mov     cx, 10d
            call    dec_div
            mov     cx, 1d
            call    dec_div
            pop     cx
            ret
bin_dec     endp
;-------------------------------------------------------------
;除法并显示输出子程序
dec_div     proc    near
            mov     ax, si
            mov     dx, 0
            div     cx
            mov     si, dx                  ;余数保存在(si)中作下一次的除法
            mov     dl, al                  ;商(在 00h~09h 范围内)送(dl)
```

```
            add     dl, 30h                 ;转换为 0~9 的 ascii 码
            mov     ah, 02h                 ;显示输出
            int     21h
            ret
dec_div     endp
;-------------------------------------------------------
;屏幕上卷
screen      proc    near                    ;入口参数为 ax
            mov     bh, 1eh                 ;设置颜色
            mov     cx, 0                   ;屏幕左上角
            mov     dx, 184fh               ;屏幕右下角
            int     10h
            ret
screen      endp
;-------------------------------------------------------
;设置光标
curs        proc    near
            mov     ah, 2                   ;设置光标
            mov     bh, 0
            mov     dh, row                 ;行号
            mov     dl, 0                   ;列号
            int     10h
            ret
curs        endp
;-------------------------------------------------------
;显示出错信息
errm        proc    near
            mov     ah, 40h                 ;向标准输出设备(文件代号=01)写文件
            mov     bx, 01                  ;标准输出设备的文件代号=01
            mov     cx, 20
            int     21h
            ret
errm        endp
;-------------------------------------------------------
disp_crlf   proc    near                    ;显示回车换行符子程序
            lea     dx, crlf
            mov     ah, 09h
            int     21h
            ret
disp_crlf   endp                            ; disp_crlf 子程序结束
; -------------------------------------------------------
            end     begin
```

11.11对 11.9 题建立的文件按下面的要求编写程序：

(1) 把所有的记录读入内存的数据缓冲区 TABLE；

(2) 显示字符串提示用户输入零(配)件号及其数量；

(3) 按零件搜索 TABLE；

(4) 如果发现所要求的零件，用它的单价计算出总价(单价×数量)；

(5) 显示零(配)件说明及总价值。

答：程序如下：

```
TITLE    READ11.EXE                     ;利用文件代号式读并计算显示程序
;Read disk records created by hancreat
;-------------------------------------------------------
            .model small
            .stack 100h
            .data
endcde      db      0                   ;结束处理指示
```

```
pathname db     'filename.lst', 0
in_mes1  db     '请输入 3 位数的零件号 Part#：', '$'
in_mes2  db     '请输入该零件的数量：', '$'
out_mes1 db     '输入的不是数字！请重新输入数字：’, '$'
out_mes2 db     '输入的零件号不存在！请重新输入 3 位数的零件号 Part#：', '$'
in_buffer db    6, ?, 6 dup(20h)            ;输入缓冲区
message  db     '       Part#       Description          Sum_Price', 0dh, 0ah,'$'
tackline db     '    |    $'
sum_price dw    0, 0
decimal  db     10 DUP(0), ‘$’
crlf     db     0dh, 0ah, '$'
handle   dw     ?
table    db     19*100 DUP(' ')             ;table，足够大
errcde   db     0                           ;错误处理指示
opnmsg   db     '***open error***', 0dh, 0ah
readmsg  db     '***read error***', 0dh, 0ah
;-------------------------------------------------
         .code
begin    proc   far
         mov    ax, @data
         mov    ds, ax
         mov    es, ax

         mov    ax, 0600h
         call   screen                      ;清屏
         call   curs                        ;设置光标
         call   openh                       ;打开文件，设置 TABLE
         cmp    endcde, 0                   ;打开错误吗？
         jnz    a0                          ;错误，转结束
         call   readh                       ;读磁盘记录
         cmp    endcde, 0                   ;读错误吗？
         jnz    a0                          ;错误，转结束
         call   in_Part                     ;没错，输入零件号和零件数量
a0:      mov    ax, 4c00h                   ;退出程序，返回 DOS
         int    21h
begin    endp
;-------------------------------------------------
;打开文件
openh    proc   near
         mov    ah, 3dh
         mov    al, 0
         lea    dx, pathname
         int    21h
         jc     bbb                         ;打开错误吗？
         mov    handle, ax                  ;没有错，保存文件代号
         ret
bbb:     mov    endcde, 01                  ;打开错误，指示结束处理
         lea    dx, opnmsg
         call   errm                        ;显示出错信息
         ret
openh    endp
;-------------------------------------------------
;读磁盘记录
readh    proc   near
         mov    ah, 3fh
         mov    bx, handle
```

```
                mov     cx, 19*100              ;准备读入的字节数
                lea     dx, table
                int     21h
                jc      c1                      ;读错误吗？
                cmp     ax, 0                   ;文件已读完吗？
                je      c2                      ;读完，退出
                cmp     table, 1ah              ;文件结束符吗?
                Je      c2
                mov     bp, ax                  ;读成功则在 AX 中返回实际读入的字节数存入 bp
                ret
c1:             lea     dx, readmsg             ;读错误
                call    errm                    ;显示出错信息
c2:             mov     endcde, 01              ;读错误或文件读完，指示结束处理
                ret
readh           endp
;-------------------------------------------------
;输入零件号和零件数量
in_Part         proc    near
                lea     dx, in_mes1             ;显示提示信息，提示输入零件号
in_Part1:       call    input                   ;输入数据
                cmp     in_buffer+1, 3          ;输入的零件号个数是 3 位吗?
                lea     dx, out_mes2            ;显示提示信息，提示重新输入零件号
                jne     in_Part1
                cld
                mov     ax, bp                  ;取实际读入文件的字节数
                mov     cl, 19                  ;每个记录的长度为 19 个字符
                div     cl                      ;计算实际读取的记录数在 al 中
                mov     bl, al
                mov     bh, 0                   ;从第 0 个记录开始顺序查找
in_Part2:       lea     si, in_buffer+2         ;查找零件号对应的零件
                lea     di, table
                mov     al, 19
                mul     bh
                add     di, ax                  ;计算某个记录的首地址
                mov     word ptr decimal, di    ;保存首地址
                mov     cx, 3
                repe    cmpsb
                je      in_Part3                ;找到对应的零件
                inc     bh                      ;找下一个记录
                cmp     bh, bl
                jb      in_Part2
                jmp     in_Part1                ;未找到对应的零件重新输入
in_Part3:       lea     dx, in_mes2             ;显示提示信息，提示输入零件数量
                call    input                   ;输入数据
                call    dec_bin                 ;将输入数据转换为二进制数，在 bx 中
                mov     di, word ptr decima     ;di 指向该记录的首地址
                mov     ax, [di+15]             ;取单价
                mul     bx                      ;总价格在(dx),(ax)中
                mov     sum_price, ax
                mov     sum_price+2, dx
                call    disp_rec                ;显示信息
                ret
in_Part         endp
;-------------------------------------------------
;输入数据
```

```
input   proc  near
input1: mov   ah, 09h                ;显示字符串
        int   21h
        mov   ah, 0ah                ;输入字符串
        lea   dx, in_buffer
        int   21h
        lea   dx, out_mes1           ;显示提示信息
        mov   cl, in_buffer+1
        cmp   cl, 0                  ;输入的数字个数为 0 吗?
        jz    input1
        mov   ch, 0
        mov   bx, 2
input2: mov   al, in_buffer[bx]      ;输入的是数字 0~9 吗?
        cmp   al, '0'
        jb    input1
        cmp   al, '9'
        ja    input1
        inc   bx
        loop  input2
        ret
input   endp
;-------------------------------------------------------
;将十进制数转换为二进制数子程序
dec_bin proc  near
        mov   bx, 0
        mov   si, 2
        mov   cl, in_buffer+1
        mov   ch, 0
transfer: mov al, in_buffer[si]      ;从十进制的高位到低位取数
        and   al, 0fh                ;将 ascii 码转换为十进制数
        mov   ah, 0
        push  cx
        xchg  ax, bx                 ;十进制数高位×10+低位 = 二进制数
        mov   cx, 10
        mul   cx
        add   bx, ax                 ;转换的二进制数在(bx)中
        pop   cx
        inc   si
        loop  transfer
        ret
dec_bin endp
;-------------------------------------------------------
;显示记录
disp_rec proc near
        call  disp_crlf
        lea   dx, message            ;显示标题
        mov   ah, 09h
        int   21h
        lea   dx, tackline           ;显示输出“   |   ”
        mov   ah, 09h
        int   21h
        mov   ah, 40h                ;向标准输出设备(文件代号=01)写文件
        mov   bx, 01                 ;标准输出设备的文件代号=01
        mov   cx, 3                  ;显示 3 位数的零件号
        mov   dx, word ptr decima    ;dx 指向该记录的首地址
        int   21h
        lea   dx, tackline           ;显示输出“   |   ”
```

```
                mov     ah, 09h
                int     21h
                mov     ah, 40h                 ;向标准输出设备(文件代号=01)写文件
                mov     bx, 01                  ;标准输出设备的文件代号=01
                mov     cx, 12                  ;显示 12 位的零件说明
                mov     dx, word ptr decima     ;dx 指向该记录的首地址
                add     dx, 3
                int     21h
                lea     dx, tackline            ;显示输出“    |    ”
                mov     ah, 09h
                int     21h
                call    bin_dec                     ;总价格转换为十进制数显示
                lea     dx, tackline            ;显示输出“    |    ”
                mov     ah, 09h
                int     21h
                call    disp_crlf
                ret
disp_rec        endp
;-------------------------------------------------------
;4 字节二进制数转换为 10 进制子程序
bin_dec         proc    near
                mov     bx, 0                   ;10 字节的 bcd 码单元清 0
                mov     cx, 10
bin_dec1:       mov     decimal[bx], 0
                inc     bx
                loop    bin_dec1
                mov     cx, 4*8                 ;4 字节二进制数共 4*8=32 位
bin_dec2:       mov     bx, 10-1                ;计算(((a31*2+a30)*2+a29)...)*2+a0
                shl     word ptr [sum_price],1      ;4 字节二进制数左移 1 位
                rcl     word ptr [sum_price +2],1
                push    cx
                mov     cx, 10
bin_dec3:       mov     al, decimal[bx]         ;计算(...)*2+ai，ai 由进位位带入
                adc     al, al
                aaa                             ;非压缩 bcd 码加法调整
                mov     decimal[bx], al
                dec     bx
                loop    bin_dec3
                pop     cx
                loop    bin_dec2
                call    disp
                ret
bin_dec         endp
;-------------------------------------------------------------
disp            proc    near                    ;显示输出子程序
                mov     cx, 10
                mov     bx, 0
disp1:          add     decimal[bx], 30h        ;变为 ascii 码
                inc     bx
                loop    disp1
                mov     cx, 10                  ;下面 5 条指令是为了不显示数据左边的“0”
                cld
                lea     di, decimal
                mov     al, 30h                 ;30h 为“0”的 ascii 码
                repe    scasb
                dec     di
                mov     dx, di
```

```
        mov   ah, 09h
        int   21h
        ret
disp    endp                          ;disp 子程序结束
;-------------------------------------------------------------
;屏幕上卷
screen  proc  near                    ;入口参数为 ax
        mov   bh, 1eh                 ;设置颜色
        mov   cx, 0                   ;屏幕左上角
        mov   dx, 184fh               ;屏幕右下角
        int   10h
        ret
screen  endp
;-------------------------------------------------
;设置光标
curs    proc  near
        mov   ah, 2                   ;设置光标
        mov   bh, 0
        mov   dh, 0                   ;行号
        mov   dl, 0                   ;列号
        int   10h
        ret
curs    endp
;-------------------------------------------------
;显示出错信息
errm    proc  near
        mov   ah, 40h                 ;向标准输出设备(文件代号=01)写文件
        mov   bx, 01                  ;标准输出设备的文件代号=01
        mov   cx, 20
        int   21h
        ret
errm    endp
;-------------------------------------------------
disp_crlf proc near                   ;显示回车换行符子程序
        lea   dx, crlf
        mov   ah, 09h
        int   21h
        ret
disp_crlf endp                        ; disp_crlf 子程序结束
; -----------------------------------------------
        end   begin
```

11.12用随机处理记录的方式编写程序，将用户需要的零(配)件记录读取到 TABLE，并根据键入的数量，计算出总价值，然后显示出零(配)件说明及总价值。

答：程序如下：

```
TITLE   READ_RAN.EXE                  ;利用文件代号式随机读并计算显示程序
;Read disk records created by hancreat
;-------------------------------------------------------
        .model small
        .stack 100h
        .data
endcde  db    0                       ;结束处理指示
pathname db   'filename.lst', 0
in_mes1 db    '请输入 3 位数的零件号 Part#：', '$'
in_mes2 db    '请输入该零件的数量：', '$'
out_mes1 db   '输入的不是数字！请重新输入数字：', '$'
out_mes2 db   '输入的零件号不存在！请重新输入 3 位数的零件号 Part#：', '$'
```

```
in_buffer db    6, ?, 6 dup(20h)            ;输入缓冲区
message   db    '       Part#       Description         Sum_Price', 0dh, 0ah,'$'
tackline  db    '    |    $'
sum_price dw    0, 0
decimal   db    10 DUP(0), '$'
crlf      db    0dh, 0ah, '$'
handle    dw    ?
table     db    19 DUP(' ')               ;table
errcde    db    0                         ;错误处理指示
opnmsg    db    '***open error***', 0dh, 0ah
readmsg   db    '***read error***', 0dh, 0ah
movmsg    db    '***move error***', 0dh, 0ah
;-------------------------------------------------
          .code
begin     proc  far
          mov   ax, @data
          mov   ds, ax
          mov   es, ax

          mov   ax, 0600h
          call  screen                    ;清屏
          call  curs                      ;设置光标
          call  openh                     ;打开文件，设置 TABLE
          cmp   endcde, 0                 ;打开错误吗？
          jnz   a0                        ;错误，转结束
          call  in_Part                   ;没错，输入零件号和零件数量
a0:       mov   ax, 4c00h                 ;退出程序，返回 DOS
          int   21h
begin     endp
;-------------------------------------------------
;打开文件
openh     proc  near
          mov   ah, 3dh
          mov   al, 0
          lea   dx, pathname
          int   21h
          jc    bbb                       ;打开错误吗？
          mov   handle, ax                ;没有错，保存文件代号
          ret
bbb:      mov   endcde, 01                ;打开错误，指示结束处理
          lea   dx, opnmsg
          call  errm                      ;显示出错信息
          ret
openh     endp
;-------------------------------------------------
;读磁盘记录
readh     proc  near
          mov   ah, 3fh
          mov   bx, handle
          mov   cx, 19                    ;准备读入的字节数
          lea   dx, table
          int   21h
          jc    c1                        ;读错误吗？
          cmp   ax, 0                     ;文件已读完吗？
          je    c2                        ;读完，退出
          cmp   table, 1ah                ;文件结束符吗?
          Je    c2
```

```
            mov   bp, ax                ;读成功则在 AX 中返回实际读入的字节数存入 bp
            ret
c1:         mov   endcde, 01            ;读错误或文件读完，指示结束处理
            lea   dx, readmsg           ;读错误
            call  errm                  ;显示出错信息
            jmp   c3
c2:         mov   endcde, 02            ;读错误或文件读完，指示结束处理
c3:         ret
readh       endp
;-------------------------------------------------
;绝对移动文件读写指针
mov_pointer     proc    near
            mov   ah, 42h
            mov   al, 0
            mov   bx, handle
            int   21h
            jc    d1                    ;错误吗?
            ret
d1:         lea   dx, movmsg            ;错误
            call  errm                  ;显示出错信息
            mov   endcde, 01            ;错误，指示结束处理
            ret
mov_pointer     endp
;-------------------------------------------------
;输入零件号和零件数量
in_Part     proc  near
            lea   dx, in_mes1           ;显示提示信息，提示输入零件号
in_Part1:   call  input                 ;输入数据
            cmp   in_buffer+1, 3        ;输入的零件号个数是 3 位吗?
            lea   dx, out_mes2          ;显示提示信息，提示重新输入零件号
            jne   in_Part1
            cld
            mov   cx, 0                 ;位移量的高位字
            mov   dx, 0                 ;位移量的低位字
            call  mov_pointer           ;绝对移动文件读写指针到文件首
in_Part2:   call  readh                 ;读磁盘记录
            cmp   endcde, 2             ;读文件结束吗?
            je    in_Part1              ;结束，未找到对应的零件重新输入
            cmp   endcde, 1             ;读错误吗?
            je    in_Part4              ;错误，转结束
            lea   si, in_buffer+2       ;查找零件号对应的零件
            lea   di, table
            mov   cx, 3
            repe  cmpsb
            je    in_Part3              ;找到对应的零件
            jmp   in_Part2              ;找下一个零件

in_Part3:   lea   dx, in_mes2           ;显示提示信息，提示输入零件数量
            call  input                 ;输入数据
            call  dec_bin                   ;将输入数据转换为二进制数，在 bx 中
            lea   di, table                 ;di 指向该记录的首地址
            mov   ax, [di+15]           ;取单价
            mul   bx                    ;总价格在(dx),(ax)中
            mov   sum_price, ax
            mov   sum_price+2, dx
```

```
            call    disp_rec                ;显示信息
in_Part4:   ret
in_Part     endp
;-------------------------------------------------------
;输入数据
input       proc    near
input1:     mov     ah, 09h                 ;显示字符串
            int     21h
            mov     ah, 0ah                 ;输入字符串
            lea     dx, in_buffer
            int     21h
            lea     dx, out_mes1            ;显示提示信息
            mov     cl, in_buffer+1
            cmp     cl, 0                   ;输入的数字个数为 0 吗?
            jz      input1
            mov     ch, 0
            mov     bx, 2
input2:     mov     al, in_buffer[bx]       ;输入的是数字 0~9 吗?
            cmp     al, '0'
            jb      input1
            cmp     al, '9'
            ja      input1
            inc     bx
            loop    input2
            ret
input       endp
;-------------------------------------------------------
;将十进制数转换为二进制数子程序
dec_bin     proc    near
            mov     bx, 0
            mov     si, 2
            mov     cl, in_buffer+1
            mov     ch, 0
transfer:   mov     al, in_buffer[si]       ;从十进制的高位到低位取数
            and     al, 0fh                 ;将 ascii 码转换为十进制数
            mov     ah, 0
            push    cx
            xchg    ax, bx                  ;十进制数高位×10+低位 = 二进制数
            mov     cx, 10
            mul     cx
            add     bx, ax                  ;转换的二进制数在(bx)中
            pop     cx
            inc     si
            loop    transfer
            ret
dec_bin     endp
;-------------------------------------------------------
;显示记录
disp_rec    proc    near
            call    disp_crlf
            lea     dx, message             ;显示标题
            mov     ah, 09h
            int     21h
            lea     dx, tackline            ;显示输出“   |   ”
            mov     ah, 09h
            int     21h
            mov     ah, 40h                 ;向标准输出设备(文件代号=01)写文件
```

```
            mov   bx, 01                ;标准输出设备的文件代号=01
            mov   cx, 3                 ;显示 3 位数的零件号
            lea   dx, table             ;dx 指向该记录的首地址
            int   21h
            lea   dx, tackline          ;显示输出“   |   ”
            mov   ah, 09h
            int   21h
            mov   ah, 40h               ;向标准输出设备(文件代号=01)写文件
            mov   bx, 01                ;标准输出设备的文件代号=01
            mov   cx, 12                ;显示 12 位的零件说明
            lea   dx, table             ;dx 指向该记录的首地址
            add   dx, 3
            int   21h
            lea   dx, tackline          ;显示输出“   |   ”
            mov   ah, 09h
            int   21h
            call  bin_dec                   ;总价格转换为十进制数显示
            lea   dx, tackline          ;显示输出“   |   ”
            mov   ah, 09h
            int   21h
            call  disp_crlf
            ret
disp_rec    endp
;-------------------------------------------------------
;4 字节二进制数转换为 10 进制子程序
bin_dec     proc  near
            mov   bx, 0                 ;10 字节的 bcd 码单元清 0
            mov   cx, 10
bin_dec1:   mov   decimal[bx], 0
            inc   bx
            loop  bin_dec1
            mov   cx, 4*8               ;4 字节二进制数共 4*8=32 位
bin_dec2:   mov   bx, 10-1              ;计算(((a31*2+a30)*2+a29)...)*2+a0
            shl   word ptr [sum_price],1        ;4 字节二进制数左移 1 位
            rcl   word ptr [sum_price +2],1
            push  cx
            mov   cx, 10
bin_dec3:   mov   al, decimal[bx]       ;计算(...)*2+ai，ai 由进位位带入
            adc   al, al
            aaa                         ;非压缩 bcd 码加法调整
            mov   decimal[bx], al
            dec   bx
            loop  bin_dec3
            pop   cx
            loop  bin_dec2
            call  disp
            ret
bin_dec     endp
;-------------------------------------------------------------
disp        proc  near                  ;显示输出子程序
            mov   cx, 10
            mov   bx, 0
disp1:      add   decimal[bx], 30h      ;变为 ascii 码
            inc   bx
            loop  disp1
            mov   cx, 10                ;下面 5 条指令是为了不显示数据左边的“0”
            cld
```

```
            lea     di, decimal
            mov     al, 30h                 ;30h 为“0”的 ascii 码
            repe    scasb
            dec     di
            mov     dx, di
            mov     ah, 09h
            int     21h
            ret
disp        endp                            ;disp 子程序结束
;-------------------------------------------------------------
;屏幕上卷
screen      proc    near                    ;入口参数为 ax
            mov     bh, 1eh                 ;设置颜色
            mov     cx, 0                   ;屏幕左上角
            mov     dx, 184fh               ;屏幕右下角
            int     10h
            ret
screen      endp
;-------------------------------------------------
;设置光标
curs        proc    near
            mov     ah, 2                   ;设置光标
            mov     bh, 0
            mov     dh, 0                   ;行号
            mov     dl, 0                   ;列号
            int     10h
            ret
curs        endp
;-------------------------------------------------
;显示出错信息
errm        proc    near
            mov     ah, 40h                 ;向标准输出设备(文件代号=01)写文件
            mov     bx, 01                  ;标准输出设备的文件代号=01
            mov     cx, 20
            int     21h
            ret
errm        endp
;-------------------------------------------------
disp_crlf   proc    near                    ;显示回车换行符子程序
            lea     dx, crlf
            mov     ah, 09h
            int     21h
            ret
disp_crlf   endp                            ; disp_crlf 子程序结束
; -----------------------------------------------
            end     begin
```